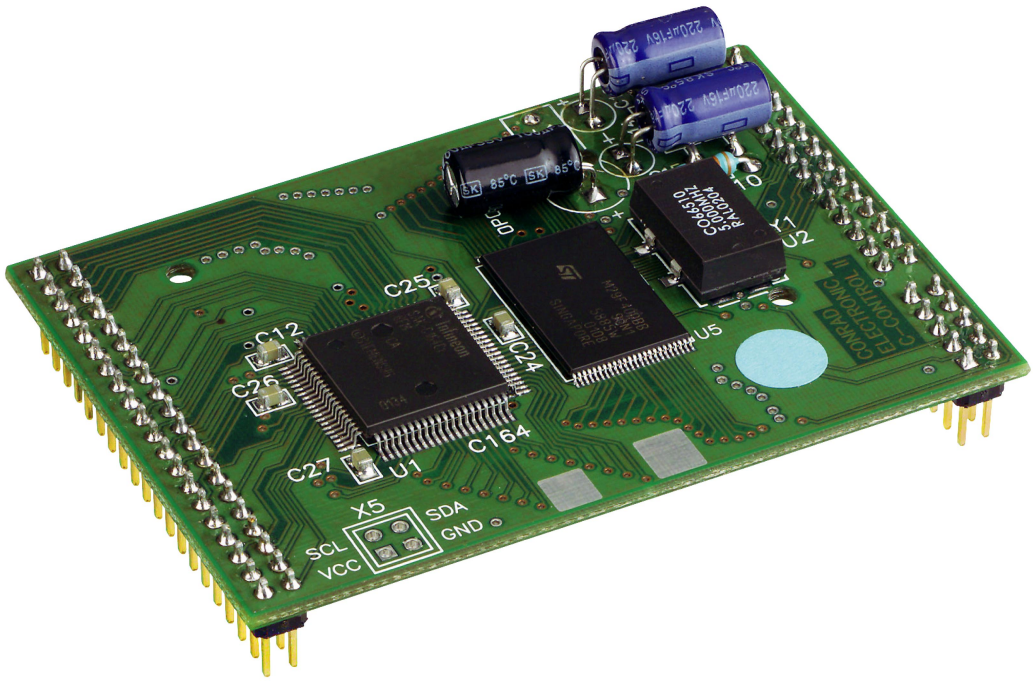
Handbuch

C-Control II Unit



Sehr geehrte Kundin, sehr geehrter Kunde,

wir danken Ihnen für Ihr Interesse und Ihr Vertrauen in unsere C-Control II Unit. Für zahllose Anwender ist C-Control bereits seit Jahren ein Begriff für kompakte, zuverlässige und preiswerte Steuerungslösungen. Neben klassischen Applikationen, wie Heizungssteuerungen und Datenerfassungssystemen, sind uns auch erfolgreiche Einsätze in der Industrieautomation, der Laborforschung oder der Midi-Technik in Tonstudios bekannt.
Vielleicht haben Sie schon mit einem unserer bewährten Systeme C-Control/BASIC, C-Control/PLUS oder der C-Control-Station gearbeitet. Eventuell sind Sie nach einiger Zeit an deren Grenzen in Bezug auf Leistungsfähigkeit und Speicherkapazität gestoßen. Oder Sie haben die genannten Systeme bisher noch nicht eingesetzt, weil sie nicht für Ihre Aufgabe geeignet schienen. Sicher kann C-Control II diese Probleme jetzt lösen!

Conrad Electronic GmbH,
Hirschau

Impressum

Diese Bedienungsanleitung ist eine Publikation der Conrad Electronic GmbH, Klaus-Conrad-Straße 1, D-92240 Hirschau.
Alle Rechte einschließlich Übersetzung vorbehalten. Reproduktionen jeder Art, z.B. Fotokopie, Mikroverfilmung oder die Erfassung in EDV-Anlagen, bedürfen der schriftlichen Genehmigung des Herausgebers.
Nachdruck, auch auszugsweise, verboten.
Diese Bedienungsanleitung entspricht dem technischen Stand bei Drucklegung, Änderung in Technik und Ausstattung vorbehalten.

© Copyright 2003 by Conrad Electronic GmbH, Printed in Germany. CTC/DT 30.10.2003

Inhaltsverzeichnis Seite

1.	**Einleitung**	**13**
2.	**Wichtige Hinweise**	**15**
2.1	Lesen dieser Anleitung	15
2.2	Handhabung	15
2.3	Bestimmungsgemäße Verwendung	16
2.4	Gewährleistung und Haftung	17
2.5	Service	18
3.	**Hardware**	**19**
3.1	(entfällt)	19
3.2	Schaltungstechnik - intern	19
3.2.1	Mikrocontroller	19
3.2.2	Speicher	20
3.2.3	Referenzspannungserzeugung	20
3.2.4	Takterzeugung	21
3.2.5	(entfällt)	21
3.3	Schaltungstechnik - extern	22
3.3.1	Anschlußbelegung - Überblick	22
3.3.2	Spannungsversorgung	24
3.3.3	(entfällt)	24
3.3.4	Reset (RSTIN, RSTOUT)	25
3.3.5	Non Maskable Interrupt (NMI)	26
3.3.6	Serielle Schnittstelle	26
3.3.7	Digitalports (P1L.0 … P1L.7, P1H.0 … P1H.7)	28
3.3.8	Sonderfunktionen der Digitalports	29
3.3.9	A/D-Ports	32
3.3.10	DCF/FRQ-Ports	32
3.3.11	PLM-Ports	33
3.3.12	CAN-Interface	34
4.	**Betriebssystem**	**36**
4.1	Überblick	36
4.2	Bootstrap - Installieren des Betriebssystems	36

4.3	Hostmodus	37
4.3.1	Systeminitialisierung und automatisches Starten von Anwenderprogrammen	37
4.3.2	Download von Anwenderprogrammen und anderen Host-Befehlen	38
4.4	Virtuelle Maschine	38
4.4.1	Grundlagen	38
4.4.2	Binärcodeinterpreter	39
4.4.3	Multithreading	40
4.4.4	Programm- und Konstantenspeicher	41
4.4.5	Datenspeicher	41
4.4.6	Stapelprozessor	42
4.4.7	Systemschnittstelle	42
5.	Die Programmiersprache C2	43
5.1	Einleitung	43
5.2	Projekte und Module	43
5.3	Syntax - Grundelemente	44
5.3.1	Kommentare	44
5.3.2	Zwischenräume	45
5.3.3	Bezeichner	45
5.3.4	Anweisungen und Anweisungsblöcke	46
5.3.5	Ausdrücke	47
5.3.6	Schlüsselworte	48
5.4	Datentypen	48
5.4.1	Numerische Datentypen	48
5.4.2	Zeichenketten (Strings)	49
5.4.3	Zusammengesetzte Datentypen	49
5.5	Variablen	50
5.5.1	Definition von Variablen	50
5.5.2	Definition und Anwendung von Variablen zusammengesetzter Datentypen	51
5.5.3	Definition und Indizierung von variablen Arrays	52
5.5.4	Initialisierung	53
5.5.5	Globale und lokale Variablen	54
5.6	Konstanten	56
5.6.1	Benannte und unbenannte Konstanten	56
5.6.2	Unbenannte Zahlenkonstanten	57
5.6.3	Unbenannte Zeichenkonstanten	57

5.6.4 Unbenannte Stringkonstanten 58
5.6.5 Definition von benannten Konstanten 59
5.6.6 Benannte konstante Arrays 60
5.7 Operatoren 61
5.7.1 Rangfolge 61
5.7.2 Arithmethische Operatoren 62
5.7.3 Bitschiebeoperatoren 62
5.7.4 Vergleichsoperatoren 63
5.7.5 Logische Operatoren und Bitmanipulationen 63
5.7.6 Stringverkettung mit dem Operator + 65
5.8 Funktionen 66
5.8.1 Funktionskopf 66
5.8.2 Parameter und lokale Variablen 68
5.8.3 Ende einer Funktion und Ergebnisrückgabe 69
5.8.4 Aufruf 70
5.8.5 Typenprüfung 71
5.8.6 Rekursion 71
5.8.7 Inline-Funktionen und -Anweisungen 72
5.9 Threads 72
5.9.1 Definition 72
5.9.2 main-Threads 73
5.9.3 Prioritätssteuerung 73
5.9.4 Warten auf Ereignisse 76
5.9.5 Pausen 76
5.9.6 Synchronisation 77
5.10 Anweisungen zur Ablaufsteuerung 82
5.10.1 if … else … - Bedingte Ausführung 82
5.10.2 loop - Endlosschleife 83
5.10.3 while - Schleife 83
5.10.4 do - Schleife 83
5.10.5 for - Schleife 84
5.10.6 Vorzeitiger Abbruch 85
5.10.7 Vorzeitige Fortsetzung 85
5.10.8 Programmende 86

6.	**Softwareentwicklung**	**87**
6.1	Installation und Start der Integrierten Entwicklungsumgebung	87
6.2	Quelltexte bearbeiten	87
6.3	Richtlinien zur Quelltextformatierung	88
6.3.1	Vorteile der einheitlichen Formatierung	88
6.3.2	Kommentare	88
6.3.3	Bezeichner	89
6.3.4	Ausdrücke	89
6.3.5	Funktionsdefinitionen	89
6.3.6	Threads	90
6.3.7	Anweisungsblöcke	90
6.3.8	Kombinationen mit Schlüsselworten zur Ablaufsteuerung	91
6.4	Automatischer Compiler	92
6.5	Simulation und Debugging	92
6.5.1	Test und Fehlersuche	92
6.5.2	Simulationsumfang	93
6.5.3	Bedienung	93
6.6	Programmübertragung in die Unit	94
7.	**Standardmodule**	**95**
7.1	can.c2 ☆☆☆	95
7.1.1	Initialisierung	95
7.1.2	Statusabfrage für einen CAN-Kanal	97
7.1.3	Test auf Übertragungsfehler	97
7.1.4	Nachricht senden	97
7.1.5	Nachricht veröffentlichen	98
7.1.6	Zählen der "Remote-Request"-Anfragen	98
7.1.7	Einstellen der Empfangs-ID	98
7.1.8	Senden einer "Remote-Request"-Anforderung	99
7.1.9	Test auf Empfang	99
7.1.10	Empfangene Daten lesen	99
7.2	hwcom.c2 und swcom.c2	99
7.2.1	Initialisierung	100
7.2.2	Einstellen der Übertragungsgeschwindigkeit	100
7.2.3	Setzen des erweiterten Empfangspuffers	100
7.2.4	Verwerfen von Daten	101

7.2.5	Test auf Empfang	101
7.2.6	Lesen eines empfangenen Bytes	101
7.2.7	Empfang von Datenrahmen	101
7.2.8	Test auf Sendebereitschaft	102
7.2.9	Senden eines Bytes	102
7.2.10	Senden von Datenrahmen	102
7.3	**i2c.c2**	**103**
7.3.1	Initialisierung	103
7.3.2	Start der Übertragung	103
7.3.3	Senden der Stopbedingung	103
7.3.4	Schreiben eines Bytes	104
7.3.5	Lesen eines Bytes mit Acknowledge	104
7.3.6	Lesen eines Bytes ohne Acknowledge	104
7.3.7	Test auf Sendebereitschaft	104
7.4	**(entfällt)**	**104**
7.4.1	(entfällt)	105
7.4.2	(entfällt)	105
7.4.3	(entfällt)	105
7.4.4	(entfällt)	105
7.4.5	(entfällt)	105
7.4.6	(entfällt)	106
7.4.7	(entfällt)	106
7.4.8	(entfällt)	106
7.4.9	(entfällt)	107
7.4.10	(entfällt)	107
7.5	**lpt.c2**	**107**
7.5.1	Initialisierung	107
7.5.2	Ausgabepuffer leeren	108
7.5.3	Test auf Ausgabebereitschaft	108
7.5.4	Ein Zeichen drucken	108
7.5.5	Text drucken	108
7.6	**math.c2**	**109**
7.6.1	Mathematische Standardfunktionen	109
7.6.2	Potenzieren	109
7.6.3	Absolutwertfunktionen	109
7.6.4	Minimum- und Maximumfunktionen	110

7.7	**mem.c2**	**110**
7.7.1	Füllen mit einem Wert	110
7.7.2	Kopieren	111
7.7.3	Speichern von Zahlenwerten in einem Bytepuffer	111
7.7.4	Lesen von Zahlenwerten aus einem Bytepuffer	112
7.8	**plm.c2**	**112**
7.8.1	Setzen der Zeitbasis	113
7.8.2	Setzen des Portmodus	113
7.8.3	Einstellen der Periodenlänge	113
7.8.4	PLM-Ausgabe	113
7.8.5	Ausgabe von Tonfrequenzen	114
7.9	**ports.c2**	**114**
7.9.1	Abfrage von Digitalports	115
7.9.2	Setzen von Digitalports	116
7.93	Umschalten und Pulsen	116
7.9.4	Deaktivieren von Ports	117
7.9.5	Pulszählung	117
7.9.6	Frequenzmessung	117
7.9.7	Analog-Digital-Wandlung	118
7.10	**str.c2**	**118**
7.10.1	String leeren	118
7.10.2	Stringlänge ermitteln	118
7.10.3	String mit Zeichen füllen	118
7.10.4	Ausgabe in einen String	119
7.10.5	Formatierte Ausgabe in einen String	119
7.10.6	Ausgabe einer Bitmaske	120
7.11	**system.c2**	**121**
7.11.1	Systemtimer	121
7.11.2	Uhrzeit	121
7.11.3	Status der DCF77-Synchonisation	122
7.11.4	Datum	122
7.11.5	Sommerzeitflag	123
7.11.6	Aufruf von Systemfunktionen ☆☆☆	123
7.11.7	Anwenderdefinierte Interruptroutinen ☆☆☆	124
7.12	**twb.c2**	**125**
7.12.1	Initialisierung	125

7.12.2 Abfrage auf Empfang des Antwortrahmens 125
7.12.3 Datenübertragung 125
7.13 constant.c2 und vmcodes.c2 126

8. Systemprogrammierung ☆☆☆ 127
8.1.1 TASKING C/C++ Tools 127
8.1.2 Ergänzungen der virtuellen Maschine und Änderungen am Betriebssystem 127
8.1.3 Implementierung eines eigenen Betriebssystem 128

9. Anhang 129
9.1 Technische Daten 129
9.1.1 Mechanik 129
9.1.2 Umgebungsbedingungen 129
9.1.3 Versorgungsspannung 129
9.1.4 Ports 129
9.2 Literaturverzeichnis 130

1 Einleitung

C-Control II ist ein kompakter Steuerungscomputer.
Das System basiert auf einem Mikrocontroller des deutschen HiTech-Unternehmens Infineon Technologies. Dieser Mikrocontroller wird z.B. in großen Stückzahlen in einigen aktuellen Fahrzeugmodellen der deutschen Automobilindustrie eingesetzt.
Dort übernimmt der Controller wichtige Steuerungsaufgaben der Bordelektronik.
C-Control II bietet Ihnen diese hochmoderne Technologie zur Lösung Ihrer Steuerungsprobleme. Sie können analoge Meßwerte und Schalterstellungen erfassen und abhängig von diesen Eingangsbedingungen entsprechende Schaltsignale ausgeben, z.B. für Relais oder Stellmotoren. In Verbindung mit einer DCF77-Funkuhraktivantenne weiß C-Control II, was die Stunde geschlagen hat und kann präzise Schaltuhrfunktionen übernehmen. Verschiedene Hardware-Schnittstellen und Bussysteme erlauben die Vernetzung von C-Control II mit Sensoren, Aktoren und anderen Steuerungssystemen.
Wir wollen unsere Technologie einem breiten Anwenderkreis zur Verfügung stellen. Aus unserer bisherigen Arbeit im C-Control-Service wissen wir, daß sich auch lernbereite Kunden ohne jegliche Elektronik- und Programmiererfahrungen für C-Control interessieren. Sollten Sie zu dieser Anwendergruppe gehören, gestatten Sie uns an dieser Stelle bitte einen Hinweis:

☞ C-Control II ist nur bedingt für den Einstieg in die Programmierung von Mikrocomputern und die elektronische Schaltungstechnik geeignet!

Wir setzen voraus, daß Sie zumindest über Grundkenntnisse in einer höheren Programmiersprache, wie z.B. BASIC, PASCAL, C, C++ oder Java verfügen. Außerdem nehmen wir an, daß Ihnen die Bedienung eines PC unter einem der Microsoft Windows Betriebssysteme (95/98/NT/2000) geläufig ist. Sie sollten auch einige Erfahrungen im Umgang mit dem Lötkolben, Multimetern und elektronischen Bauelementen haben.
Wir haben uns bemüht, alle Beschreibungen so einfach wie möglich zu formulieren. Leider können wir in einer Bedienungsanleitung zum hier vorliegenden Thema nicht immer auf den Gebrauch von Fachausdrücken und Anglizismen verzichten. Schlagen Sie diese bei Bedarf bitte in entsprechenden Fachbüchern nach.

Teile dieser Anleitung können ohne Vorankündigung ungültig werden. Das ist bedingt durch eventuell notwendige technische Änderungen zur Verbesserung des Produktes oder zur Anpassung an die aktuelle Verfügbarkeit von Bauelementen. Wir werden die dann notwendigen Informationen im Internet unter www.c-control.de bereitstellen.

In dieser neuen Version der Bedienungsanleitung der C-Control II sind einige Absätze gekürzt worden, da sich die Bauart und der Lieferumfang der C-Control geändert hat.
Diese sind in der Bedienungsanleitung mit *(entfällt)* gekennzeichnet.

Wir wünschen Ihnen viel Freude mit C-Control II.
Ihr Conrad Technologie Centrum

Conrad Electronic GmbH
Klaus-Conrad-Straße 1
92240 Hirschau

2 Wichtige Hinweise

2.1 Lesen dieser Anleitung

Bitte lesen Sie diese Anleitung, bevor Sie die C-Control II Unit in Betrieb nehmen. Während einige Kapitel nur für das Verständnis der tieferen Zusammenhänge von Interesse sind, enthalten andere wichtige Informationen, deren Nichtbeachtung zu Fehlfunktionen oder Beschädigungen führen kann. Kapitel und Absätze, die Themen mit gehobenem Schwierigkeitsgrad enthalten, sind durch das Symbol ☆☆☆ gekennzeichnet. Sie können diese Themen zu einem späteren Zeitpunkt aufgreifen, nachdem Sie erste Erfahrungen mit der Anwendung der C-Control II Unit und der Programmiersprache C2 gesammelt haben.

☞ Für Schäden, die aus der Nichtbeachtung von Hinweisen in dieser Anleitung resultieren, besteht keinerlei Gewährleistungsanspruch und übernehmen wir keine Haftung.

2.2 Handhabung

☞ Die C-Control II Unit enthält empfindliche Bauteile. Diese können durch elektrostatische Entladungen zerstört werden!

Beachten Sie die allgemeinen Regeln zur Handhabung elektronischer Bauelemente. Richten Sie Ihren Arbeitsplatz fachgerecht ein. Erden Sie Ihren Körper vor der Arbeit, z.B. durch Berühren eines geerdeten, leitenden Gegenstandes. Vermeiden Sie die Berührung der Anschlußpins der C-Control II Unit.

Entfernen Sie vor der ersten Benutzung der C-Control II Unit eventuelle Rückstände des Antistatic-Schaumstoffes von den Pins.

2.3 Bestimmungsgemäße Verwendung

Die C-Control II Unit ist ein elektronisches Bauelement im Sinne eines integrierten Schaltkreises. Die C-Control II Unit dient zur programmierbaren Ansteuerung elektrischer und elektronischer Geräte. Der Aufbau und Betrieb dieser Geräte muß konform zu geltenden europäischen Zulassungsrichtlinien (CE) erfolgen.

Die C-Control II Unit darf nicht in galvanischer Verbindung zu Spannungen über Schutzkleinspannung stehen. Die Ankoppelung an Systeme mit höherer Spannung darf ausschließlich über Komponenten mit VDE-Zulassung erfolgen. Dabei müssen die vorgeschriebenen Luft- und Kriechstrecken eingehalten sowie ausreichende Maßnahmen zum Schutz vor Berührung gefährlicher Spannungen getroffen werden.

Auf der Platine der C-Control II Unit arbeiten elektronische Bauelemente mit hochfrequenten Taktsignalen und steilen Pulsflanken. Bei unsachgemäßem Einsatz der Unit kann das zur Aussendung elektromagnetischer Störsignale führen. Die Ergreifung entsprechender Maßnahmen (z.B. Verwendung von Drosselspulen, Begrenzungswiderständen, Blockkondensatoren und Abschirmungen) zur Einhaltung gesetzlich vorgeschriebener Maximalwerte liegt in der Verantwortung des Anwenders.

Die maximal zulässige Länge angeschlossener Leitungen ohne zusätzliche Maßnahmen beträgt 0,25 Meter (Ausnahme siehe CAN-Interface und Serielle Schnittstelle).

Unter dem Einfluß von starken elektromagnetischen Wechselfeldern oder Störimpulsen kann die Funktion der C-Control II Unit beeinträchtigt werden. Gegebenenfalls sind ein Reset und ein Neustart des Systems erforderlich.

Achten Sie beim Anschluß von externen Baugruppen auf die zulässigen maximalen Strom- und Spannungswerte der einzelnen Pins. Das Anlegen einer verpolten oder zu hohen Spannung oder die Belastung mit einem zu hohen Strom kann zur sofortigen Zerstörung der Unit führen.

2.4 Gewährleistung und Haftung

Conrad Electronic bietet für die C-Control II Unit eine Gewährleistungsdauer von 24 Monaten ab Rechnungsdatum. Innerhalb dieses Zeitraums werden defekte Units kostenfrei umgetauscht, wenn der Defekt nachweislich auf einen Produktionsfehler oder Transportschaden zurückzuführen ist.

Die Software im Betriebssystem des Mikrocontrollers sowie die PC-Software auf CD-ROM werden in der vorliegenden Form geliefert. Conrad Electronic übernimmt keine Garantie dafür, daß die Leistungsmerkmale dieser Software individuellen Anforderungen genügen und daß die Software in jedem Fall unterbrechungs- und fehlerfrei arbeitet.

Conrad Electronic übernimmt keine Haftung für Schäden, die unmittelbar durch oder in Folge der Anwendung der C-Control II Unit entstehen. Der Einsatz der C-Control II Unit in Systemen, die direkt oder indirekt medizinischen, gesundheits- oder lebenssichernden Zwecken dienen, ist nicht zulässig.

Sollte die C-Control II Unit inklusive Software Ihre Ansprüche nicht befriedigen, oder sollten Sie mit den Gewährleistungs- und Haftungsbedingungen nicht einverstanden sein, nutzen Sie unsere 14tägige Geld-Zurück-Garantie. Bitte geben Sie uns die Unit dann innerhalb dieser Frist ohne Gebrauchsspuren, in unbeschädigter Originalverpackung und mit allem Zubehör zur Erstattung oder Verrechnung des Warenwertes zurück!

2.5 Service

Conrad Electronic stellt Ihnen ein Team von erfahrenen Servicemitarbeitern zur Seite. Sollten Sie Fragen zur C-Control II Unit haben, erreichen Sie unsere Technische Kundenbetreuung telefonisch, per Brief, Fax oder E-Mail.

Telefon: 0180 / 53 12 116
FAX: 0180 / 53 12 119

Postanschrift: Conrad Electronic GmbH
TKB Computer- und Meßtechnik
Klaus-Conrad-Str. 1
92240 Hirschau

E-Mail: webmaster@c-control.de

Bitte nutzen Sie vorzugsweise die Kommunikation per E-Mail. Wenn Sie ein Problem haben, geben Sie uns nach Möglichkeit eine Skizze Ihrer Anschlußschaltung als angehängte Bilddatei (im JPG-Format) sowie den auf das Problem reduzierten Teil Ihres Programmquelltextes (maximal ca. 20 Zeilen). Bitte haben Sie Verständnis dafür, daß wir keine Quelltexte analysieren, die nicht nach den in dieser Anleitung empfohlenen Stilregeln formatiert sind (siehe 6.3). Sollten Sie Interesse an einer kundenspezifischen Softwarelösung haben, vermitteln wir Ihnen gern ein entsprechendes Angebot.
Weiterführende Informationen und aktuelle Software zum Download finden Sie auf der C-Control Homepage im Internet unter www.c-control.de.

3 Hardware

In diesem Kapitel erfahren Sie die wichtigsten Grundlagen zur Hardware der C-Control II Unit. Auf den hinteren Umschlagseiten finden Sie den vollständigen Schaltplan der C-Control II Unit. Er dokumentiert den inneren Aufbau und die Funktionsweise der Unit. Die Schaltung wurde nach Applikationsvorschlägen aus den Datenblättern der verwendeten ICs entwickelt. Bei Fragen zu Details konsultieren Sie bitte diese Datenblätter. Sie finden diese im Internet als PDF-Dateien, einige davon auch auf der Utility-CD.

3.1 *(entfällt)*

3.2 Schaltungstechnik - intern

3.2.1 Mikrocontroller

Der Mikrocontroller C164CI stammt aus der C166-Familie von Infineon Technologies (früher SIEMENS Halbleiter) und ist der "kleine Bruder" des C167. Im 80poligen QFP-Gehäuse bietet der C164CI zahlreiche interessante Hardwareressourcen, wie z.B.

· acht 10bit-Analog-Digitalwandlerports
· eine CAN-Busschnittstelle
· eine asynchrone serielle Schnittstelle
· komfortable Timer

Der Mikrocontroller verarbeitet Daten mit einer Breite von 16 Bit. Sein Adreßraum umfaßt 16MB - Speicheradressen bestehen aus einem Segment-Byte und einem Offset-Word.
Die Kontrolle der umfangreichen Hardwareressouren des Controllers erfolgt über Einträge in die Special Function Registers (SFR). Diese befinden sich in einem Bereich des ersten Speichersegments im internen RAM des Controllers. Das System der C-Control II Unit kapselt die zum Teil sehr komplexen Zugriffsmechanismen auf die Hardwareressourcen in relativ einfachen Funktionsaufrufen (siehe Kapitel 7).

3.2.2 Speicher

In der C-Control II Unit sind 512kB FLASH-EEPROM (8 Segmente) und 64kB SRAM (1 Segment) an den Mikrocontroller angeschlossen. Im Schaltplan erkennen Sie die Dekodierung der Adreßbussignale und die Anschaltung der Speicher-ICs an den Controller. Intern verfügt der Controller über 64kB OTP-ROM (one time programmable - einmalig programmierbar) sowie 1kB RAM, inklusive Universal Register und Special Function Register. Das interne RAM wird vom Betriebssystem der C-Control II Unit über einen Teil des ersten FLASH-Segments gelegt. Der interne OTP-Bereich ist deaktiviert und nicht nutzbar.
Der gesamte Speicher ist vom System wie folgt aufgeteilt:

Segment	Adressen	physischer Speichertyp	Verwendung
0	0x00000…0x0FFFF	ext. FLASH-EEPROM,	Betriebssystem,
		internes RAM, Register	Hardwarezugriff
1	0x10000…0x1FFFF	ext. FLASH-EEPROM	Betriebssystemreserve
2	0x20000…0x2FFFF	ext. FLASH-EEPROM	Betriebssystemreserve
3	0x30000…0x3FFFF	ext. FLASH-EEPROM	Anwendersystemroutinen
4	0x40000…0x4FFFF	ext. FLASH-EEPROM	C2-Programm VM-Codes
5	0x50000…0x5FFFF	ext. FLASH-EEPROM	C2-Programm VM-Codes
6	0x60000…0x6FFFF	ext. FLASH-EEPROM	C2-Programm Konstanten
7	0x70000…0x7FFFF	ext. FLASH-EEPROM	C2-Programm Konstanten
8	0x80000…0x8FFFF	ext. SRAM	C2-Programm Daten

3.2.3 Referenzspannungserzeugung

Der Mikrocontroller verfügt über einen Analog-Digital-Wandler mit einer Auflösung von 10 Bit. Das heißt, gemessene Spannungen können als ganze Zahlen von 0 bis 1023 dargestellt werden. Die Referenzspannung für die untere Grenze ist der GND-Pegel, also 0V. Die Referenzspannung für die obere Grenze beträgt 4,096V und wird durch ein präzises

Referenzspannungs-IC erzeugt. Die Toleranz der Referenzspannung liegt unter einem Prozent. Die maximale Temperaturdrift über den gesamten zulässigen Betriebstemperaturbereich beträgt 50ppm (parts per million, 1ppm = 0,0001 Prozent).
Eine Differenz von einem Bit des digitalisierten Meßwertes entspricht einer Spannungsdifferenz von 4mV. Ist x ein digitaler Meßwert, dann errechnet sich der entsprechende Spannungswert u wie folgt:

u = x * 4,096V / 1024
oder einfach:
u = x * 0,004V

3.2.4 Takterzeugung

Die Takterzeugung erfolgt durch einen 5MHz-Quarzoszillator. Im Controller erzeugt eine PLL-Schaltung daraus den 20MHz-Systemtakt.

3.2.5 *(entfällt)*

3.3 Schaltungstechnik - extern

3.3.1 Anschlußbelegung - Überblick

X4

P1H.0	P1H.1
P1H.2	P1H.3
P1H.4	P1H.5
P1H.6	P1H.7
GND	GND
GND	PLM 0
BEEP	PLM 1
FRQ 1	DCF/FRQ 0
GND	GND
ADC 0	ADC 1
ADC 2	ADC 3
ADC 4	ADC 5
ADC 6	ADC 7
GND	GND
I2C SCL	I2C SDA
CAN TxD	CAN RxD
CANL	CANH
digital TxD	digital RxD
digital CTS	µC CTS
digital RTS	µC RTS

X3

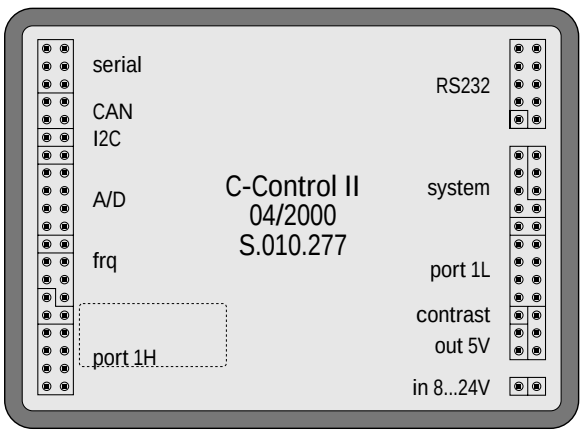
POWER	GND

X2

5 Volt	GND
5 Volt	GND
LCD ADJ	GND
P1L.7	P1L.6
P1L.5	P1L.4
P1L.3	P1L.2
P1L.1	P1L.0
GND	GND
RSTOUT	NMI
RSTIN	GND
BOOT	GND
HOST	GND

X1

GND	n.c.
n.c.	n.c.
TxD	CTS
RxD	RTS
n.c.	n.c.



Unterseite

Allgemein: GND … ground (Masse, Bezugspotential)
n.c. … not connected (nicht belegt, frei)

X1:
TxD … serielle Schnittstelle (hwcom) - Sendeleitung, NRZ-Signal
RxD … serielle Schnittstelle (hwcom) - Empfangsleitung, NRZ-Signal
RTS … serielle Schnittstelle (hwcom) - Request To Send, NRZ-Signal
CTS … serielle Schnittstelle (hwcom) - Clear To Send, NRZ-Signal

X2:
5 Volt … stabilisierte Ausgangsspannung des Schaltreglers
LCD ADJ … Eingang für LCD-Kontrasteinstellung
P1L.0 … 7 … 8 Digitalports
NMI … non-maskable interrupt
RSTOUT … RESET out
RSTIN … RESET in
BOOT … Aktivieren des Bootstrap-Modus nach Reset
HOST … Aktivieren des Hostmodus nach Reset

X3:
POWER … Versorgungsspannungseingang 8...24V

X4:
P1L.0 … 7 … 8 Digitalports
BEEP … Tonfrequenzausgabe (oder PLM 2)
PLM 0/1 … Ausgänge für Pulslängenmodulation
DCF /FRQ 0 … Funkuhrempfang oder Frequenzmessung
FRQ 1 … Frequenzmessung
ADC 0 … 7 … 8 Analog-Digital-Converter Eingänge
I^2C SCL … I^2C-Bus Taktleitung
I^2C SDA … I^2C-Bus Datenleitung
CAN_Tx … CAN-Bus, Digitalpegel des Senders
CAN_Rx … CAN-Bus, Digitalpegel des Empfängers, nur für Diagnosezwecke!
CANL/H … CAN-Bus
digital TxD … hwcom - digitale Sendeleitung
digital RxD … hwcom - digitale Empfangsleitung, nur für Diagnosezwecke!
digital RTS … hwcom - digitale RTS-Leitung
digital CTS … hwcom - digitale CTS-Leitung
µC RTS … hwcom - RTS-Port am Mikrocontroller
µC CTS … hwcom - CTS-Port am Mikrocontroller

Anmerkung zu "CAN_Rx" und "digital RxD": an diesen Pins darf kein Signal eingespeist werden! Sie dienen ausschließlich zur Signalanalyse, z.B. mit Hilfe eines Speicheroszilloskopes.

3.3.2 Spannungsversorgung (POWER, 5 Volt, GND)

In der Unit befindet sich ein kompaktes IC-Schaltnetzteil, das aus der am Pin POWER eingespeisten Versorgungsspannung die stabilisierte 5V-Betriebsspannung für den Mikrocontroller und alle anderen Schaltkreise erzeugt. Durch die Leistungsreserven des Netzteils stehen diese 5V auch zur Versorgung externer ICs zur Verfügung und können an der Unit abgegriffen werden. Beachten Sie den maximal entnehmbaren Strom (siehe Kapitel 9.1 Technische Daten). Eine Überschreitung kann zur Zerstörung der C-Control II Unit führen!

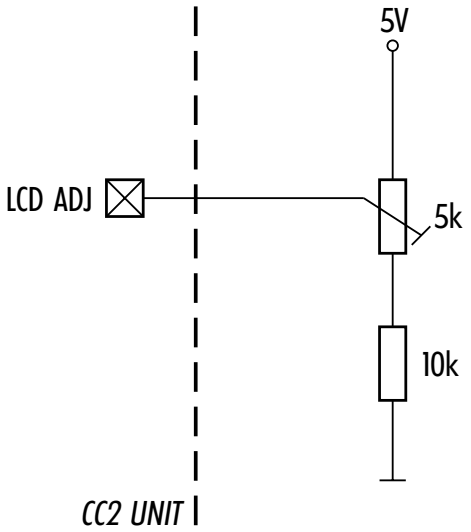
Die Gleichspannung zur Versorgung der Unit muß im Bereich von 8V bis 24V liegen. Diese Spannung kann unstabilisiert sein, darf aber nur eine geringe Restwelligkeit aufweisen. Bei Bedarf ist die Versorgungsspannung mit zusätzlichen Kondensatoren (Elko 470µF) zu stützen. In die Spannungszuleitung sollte in Reihenschaltung, nahe an der Unit, eine Drosselspule eingefügt werden, um die leitungsgebundene Störaussendung zu unterdrücken.

Wegen der relativ hohen Stromaufnahme der C-Control II Unit im Bereich von 50…100mA ist sie für den Einsatz in dauerhaft batteriebetriebenen Geräten nicht zu empfehlen.

Beachten Sie den Hinweis zu kurzzeitigen Ausfällen der Versorgungsspannung in Kapitel 3.3.4.

3.3.3 LCD-Kontrast (LCD ADJ)

Die beste Sichtbarkeit der LCD-Zeichen ergibt sich bei frontaler Betrachtung. Gegebenenfalls muß der Kontrast etwas nachgeregelt werden. Das kann über eine Widerstandsschaltung am Pin LCD ADJ erfolgen.

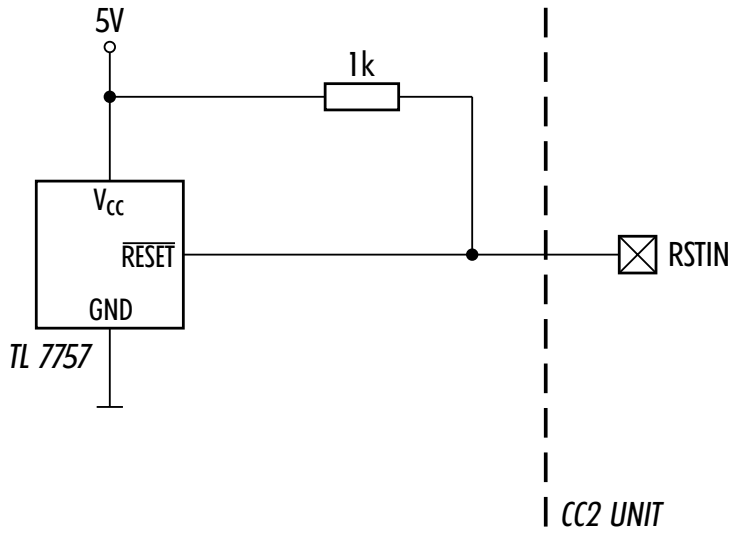


3.3.4 Reset (RSTIN, RSTOUT)

Ein Reset bewirkt die Rückkehr des Mikrocontrollersystems in einen definierten Anfangszustand. Die C-Control II Unit kennt grundsätzlich 3 Reset-Quellen

- Power-On-Reset: wird automatisch nach dem Einschalten der Betriebsspannung ausgeführt.
- Hardware-Reset: wird ausgeführt wenn der RSTIN-Pin der Unit "low" gezogen und wieder losgelassen wird, z.B. durch kurzes Drücken eines angeschlossenen Reset-Tasters. Werden am RSTIN-Pin die Ausgänge externer ICs oder Schaltungen angeschlossen, so müssen diese Open-Collector bzw. Open-Drain-Charakter haben!
- Software-Reset: Über einen Assembler-Befehl ("SRST") kann softwaremäßig ein Reset ausgelöst werden.

Bitte beachten Sie: Kurzzeitige Ausfälle der Versorgungsspannung führen zu folgendem Effekt: die 5V-Betriebsspannung des Controllers fällt gestützt durch Elektrolytkondensatoren langsam ab. Sie steigt jedoch wieder auf 5V, bevor die Kondensatoren ganz entladen sind ("Brown-Out"-Effekt). Dadurch unterbricht der Mikrocontroller seine Arbeit und erhält keinen korrekten Power-On-Reset zum Neustart.
Sollte in einer Applikation die Gefahr kurzzeitiger Ausfälle der Versorgungsspannung bestehen, muß eine Spannungswächterschaltung an die C-Control II Unit angeschlossen werden, z.B. mit Hilfe des ICs TL 7757:



RSTOUT ist ein Ausgangssignal des Mikrocontrollers. Es ändert seinen Pegel nach "low", wenn eine Restbedingung eintritt. Es wird vom Betriebssystem erst nach Abschluß einiger Systeminitialisierungen wieder auf "high" gesetzt. Somit können externe Schaltungen rückgesetzt werden, die erst arbeiten dürfen, wenn C-Control II Unit nach dem Reset ihre Grundeinstellungen vorgenommen hat. Der RSTOUT-Pin sollte ohne interne Kenntnis des Betriebssystems nicht benutzt werden.

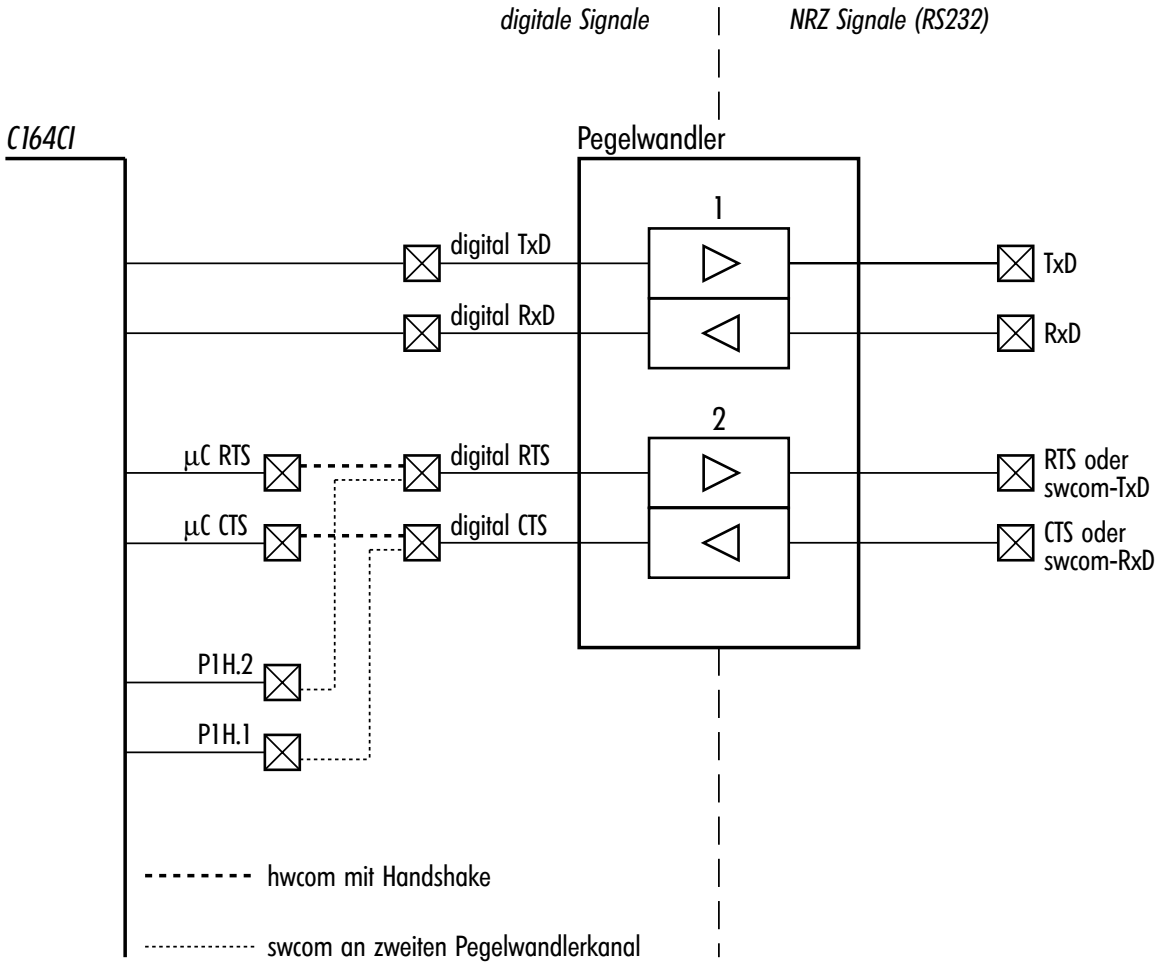
3.3.5 Non Maskable Interrupt (NMI)

Dieser Anschluß wird von der aktuellen Systemversion nicht benutzt. Schließen Sie ohne interne Kenntnis des Betriebssystems keine externen Bauteile daran an.

3.3.6 Serielle Schnittstelle

Der Mikrocontroller C164CI besitzt hardwareseitig eine asynchrone serielle Schnittstelle nach RS232-Standard (= erste serielle Schnittstelle, ="hwcom"). Eine zweite asynchrone serielle Schnittstelle kann vom Betriebssystem per Software an den Digitalports P1H.1 (Empfangen) und P1H.2 (Senden) emuliert werden. (= zweite serielle Schnittstelle, ="swcom"). Das Betriebssystem setzt für beide Schnittstellen das Format 8-N-1 fest, also jedes Byte wird mit 8 Datenbits, ohne Paritätsbit und mit einem Stopbit übertragen. Andere Formate werden von C-Control II nicht unterstützt. Bei entsprechenden Detailkenntnissen der Programmierung des C164CI-Mikrocontrollers können per Systemprogrammierung auch andere Formate realisiert werden (siehe Kapitel 8).
Auf der Platine der C-Control II Unit befindet sich ein hochwertiges Pegelwandler-IC zur Umsetzung der digitalen Bitströme in Non-Return-Zero-Signale nach dem RS232-Standard (positive Spannung für Lowbits, negative Spannung für Highbits). Das IC hat jeweils zwei Kanäle in jeder Wandlerrichtung. Das Pegelwandler-IC verfügt über einen erhöhten Schutz vor Spannungsspitzen. Spannungsspitzen können in elektromagnetisch belastetem Umfeld, z.B. in industriellen Anwendungen, in die Schnittstellenkabel induziert werden und angeschlossene Schaltkreise zerstören.
Ein Kanal ist fest mit dem digitalen seriellen Ein-/Ausgang des Mikrocontrollers verbunden. Diese digitalen Signale können an den Pins "digital TxD" und "digital RxD" der Unit beobachtet werden. Die NRZ-Seite des ersten Kanals des Pegelwandlers finden Sie an den Pins TxD und RxD. Im Ruhezustand (keine aktive Datenübertragung) können Sie am Pin TxD eine negative Spannung von einigen Volt gegen GND messen. RxD ist hochohmig. Das der C-Control II Unit beiliegende Nullmodem-Kabel kreuzt die Leitungen RxD und TxD und verbindet sie mit den Pins 3 (PC TxD) bzw. 2 (PC RxD) der 9poligen SUB-D Buchse des PCs. Außerdem wird eine Masseverbindung zwischen GND der Unit und Pin 5 der 9poli-

gen SUB-D Buchse des PCs hergestellt.
Der zweite Kanal des Pegelwandlers kann entweder für RTS/CTS-Handshake-Signale für die erste serielle Schnittstelle (hwcom) oder für die Datenleitungen der zweiten seriellen Schnittstelle (swcom) verwendet werden. Die NRZ-Seite des zweiten Kanals finden Sie an den Pins RTS und CTS. Wenn Sie die Pins "digital RTS" mit "µC RTS" und "digital CTS" mit "µC CTS" verbinden, dann arbeitet die erste serielle Schnittstelle (hwcom) mit Handshake-Signalen. Alternativ können Sie die Pins "digital RTS" mit P1H.2 und "digital CTS" mit P1H.1 verbinden. Dann ist die zweite serielle Schnittstelle (swcom) am Pegelwandler angeschlossen.



Eine Kabelverbindung mit Anschluß an die NRZ-Pins TxD, RxD, RTS, CTS darf bis zu 10 Metern lang sein. Verwenden Sie nach Möglichkeit geschirmte Normkabel. Bei längeren Leitungen oder ungeschirmten Kabeln können Störeinflüsse die Datenübertragung beeinträchtigen. Verwenden Sie nur Verbindungskabel, deren Anschlußbelegung Sie kennen.

☞ Verbinden Sie niemals die seriellen Sendeausgänge zweier Geräte miteinander! Sie erkennen die Sendeausgänge in der Regel an der negativen Ausgangsspannung im Ruhezustand.

☞ Wird die zweite serielle Schnittstelle ohne Pegelwandler betrieben, gilt wie für alle digitalen Ports eine maximal zulässige Leitungslänge von 0,25 Metern.

☞ Wird die zweite serielle Schnittstelle mit dem Pegelwandler betrieben, darf die C-Control II Unit nicht mehr direkt von der 10poligen Steckerleiste X1 (siehe 3.3.1 Anschlußbelegung), über den Schnittstellenadapter und das beiliegende Nullmodemkabel mit einem PC verbunden werden.

3.3.7 Digitalports (P1L.0 … P1L.7, P1H.0 … P1H.7)

Die C-Control II Unit führt 16 digitale Ports des Mikrocontrollers nach außen: P1L.0 … P1L.7 sowie P1H.0 … P1L.7. An den Digitalports können z.B. Taster mit Pull-Up-Widerständen, Digital-ICs, Optokoppler oder Treiberschaltungen für Relais angeschlossen werden. Die Ports können einzeln, in Vierergruppen ("Nibble"), byteweise oder im ganzen als 16bit-Word angesprochen werden (siehe Kapitel 7.9).
Ein Port ist entweder Eingang oder Ausgang.

☞ Schalten Sie niemals zwei Ports direkt zusammen, die gleichzeitig als Ausgang arbeiten sollen!

Eingangsports sind hochohmig und überführen ein anliegendes Spannungssignal in einen logischen Wert. Voraussetzung dafür ist, daß sich das Spannungssignal innerhalb der für TTL- bzw. CMOS-ICs definierten Bereiche für Low- oder Highpegel befindet. In der weiteren Verarbeitung im Programm werden die logischen Werte von einzelnen Eingangsports als 0 ("low") oder -1 ("high") dargestellt. Nibbleports nehmen Werte von 0 bis 15 an, Byteports 0 bis 255. Der Wordport wird entsprechend dem 16bit-Integerzahlenbereich als ein Wert von -32768 bis 32767 gelesen.
Ausgangsports können über eine interne Treiberschaltung digitale Spannungssignale ausgeben. Angeschlossene Schaltungen können einen geringen Strom aus den Ports ziehen (bei High-Pegel) bzw. in diesen speisen (bei Low-Pegel). Beachten Sie den maximal zulässigen Laststrom für einen einzelnen Port und für alle Ports in Summe (siehe Kapitel 9.1 Technische Daten). Eine Überschreitung der Maximalwerte kann zur Zerstörung der C-Control II Unit führen.

Nach dem Reset ist zunächst jeder Digitalport als Eingangsport konfiguriert. Ein Port wird automatisch zum Ausgangsport, wenn das Anwenderprogramm einen Ausgabewert an diesen schreibt. Durch Aufruf einer speziellen Funktion der Standardmodule kann ein Ausgangsport jedoch wieder deaktiviert werden, d.h. in den hochohmigen Zustand gebracht werden (7.9.4).

3.3.8 Sonderfunktionen der Digitalports

Einige Digitalports stehen alternativ für spezielle Ein-/Ausgabe-Operationen zur Verfügung. Dazu müssen zunächst die entsprechenden Initialisierungsfunktionen der Systemmodule aufgerufen werden (siehe z.B. Kapitel 7.2, 7.5, 7.12). Beachten Sie, daß der Aufruf einer Initialisierung alle konkurrierenden Portfunktionen deaktiviert.

Pin	Standardfunktion	Sonderfunktion
P1L.0…7	Digital-Byteport	Datenbits der Druckerschnittstelle
P1H.0	Digital-, Counter-und Interruptport	-
P1H.1	Digital-, Counter-und Interruptport	swcom RxD
P1H.2	Digital-, Counter-und Interruptport	swcom TxD
P1H.3	Digital-, Counter-und Interruptport	clock für Anschluß eines 2W-Bus-Modems
P1H.4	Digitalport	data für Anschluß eines 2W-Bus-Modems
P1H.5	Digitalport	BUSY (Handshake) der Druckerschnittstelle
P1H.6	Digitalport	ACKNOWLEDGE (Handshake) der Druckerschnittstelle
P1H.7	Digitalport	STROBE (Handshake) der Druckerschnittstelle

3.3.8.1 Zähler und Interruptports

Die vier Ports P1H.0 … P1H.3 der C-Control II Unit sind interruptsensibel. Nach dem Reset sind sie vom Betriebssystem wie folgt konfiguriert: Bei jeder High-Low-Flanke an einem der Pins wird in eine von vier Systeminterruptroutinen verzweigt. In dieser Routine wird einer von vier Zählerwerten um 1 erhöht. Außerdem prüft das System, ob eine besondere Behandlungsroutine des Anwenders installiert ist und führt diese gegebenenfalls aus (siehe Kapitel 7.11.7 und 8.1.2).

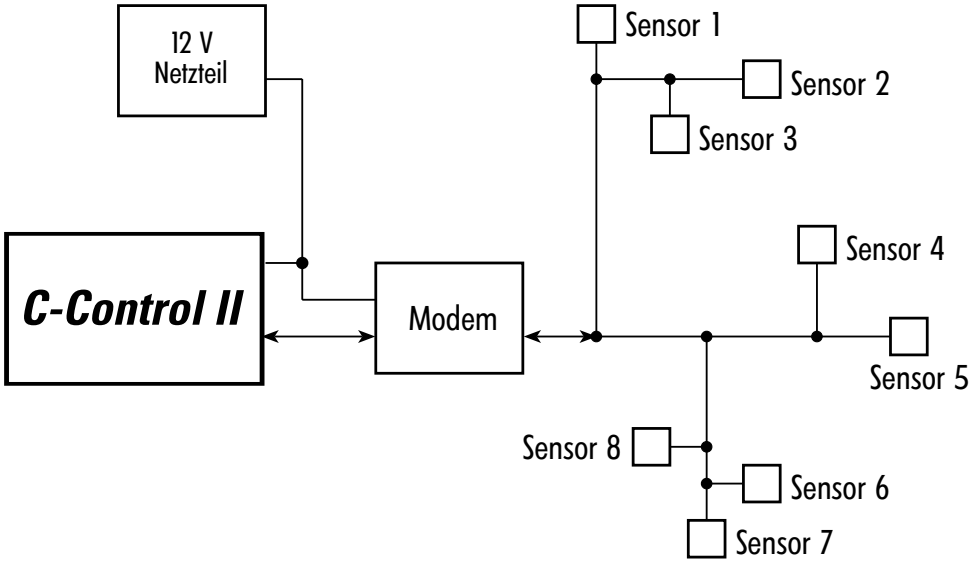
Die Zählereingänge können Pulse mit Abständen bis hinab zu ca. einer Millisekunde verlustfrei zählen. Sollte eine kürzere Reaktionszeit notwendig sein, kann das mit Hilfe der Systemprogrammierung durch Erhöhen der Interruptprioritäten erfolgen.

3.3.8.2 Zweite serielle Schnittstelle (swcom)

An den Digitalports P1H.1 und P1H.2 kann das Betriebssystem softwaremäßig eine zweite asynchrone Schnittstelle emulieren. Lesen Sie dazu die Kapitel 3.3.6 und 7.2.

3.3.8.3 Zweidrahtbus

Conrad Electronic hat eine Familie von Sensor- und Aktuatormodulen entwickelt, mit denen ein Steuercomputer, wie die C-Control II Unit, um zusätzliche Ein- und Ausgabefunktionen erweitert werden kann. Die Besonderheit dieser Module ist die einfache Vernetzung über ein Zweidrahtbussystem (englisch "two wire bus", abgekürzt auch "2W-Bus", "2WB" oder "TWB"). Bei diesem Bussystem werden digitale Daten über eine 12V-Gleichspannungsleitung übertragen. Diese Leitung übernimmt gleichzeitig die Versorgung der Module. Die Busstruktur ist eine Baumstruktur.



Zur Zeit stehen folgende 2W-Bus-Module zur Verfügung:

- Digitalportmodul (4 zusätzliche I/O-Ports)
- Digitalportmodul mit Leistungsausgängen (4 Ports bis max 2A / 25V)
- Frequenzmesser- / Zählermodul (1 Eingang, bis 30kHz-Pulsfrequenz, integrierter Reed-Kontakt zur Triggerung mit einem externen Magneten, z.B. zur Überwachung von Türen und Fenstern)
- Kombimodul: 1 x A/D-Wandler (10bit, 0 ... 2,5V) und 2 x digital I/O
- Kombimodul: 1 x Temperatursensor -23°C...100°C (0.125K Auflösung) und 2 x digital I/O
- Infrarotsender/-empfänger (zur Fernsteuerung von Geräten durch C-Control oder zur Fernbedienungen von C-Control-Applikationen durch Infrarot)
- Minidisplay (für zusätzliche Anzeigen im Stil des Minidisplays der C-Control II Unit)

Zur Ankoppelung der C-Control II Unit an den 2W-Bus benötigen Sie das 2W-Bus-Modem. Dieses Modem hat einen Eingang für die unmodulierte 12V-Versorgungsspannung aus einem Netzteil sowie eine synchrone digitale Schnittstelle mit einer Daten-, einer Takt- und einer Masseleitung (DATA, CLOCK, GND). Ausgangsseitig befindet sich der 2W-Bus-Anschluß. Die Leitungslänge vom 2W-Bus-Modem zu einem 2W-Bus-Modul kann bis zu 20m betragen. Die Verbindung der C-Control II Unit mit dem 2W-Bus-Modem erfolgt über die Pins P1H.3 (CLOCK) und P1H.4 (DATA) sowie GND.
Im Betrieb sendet die C-Control II Unit 8 Byte lange Datenrahmen seriell-synchron an das 2W-Bus-Modem. Diese Rahmen enthalten die Adresse des angesprochenen 2W-Bus-Moduls, ein Kommando und einige Datenbytes. Nach einer kurzen Zeit (ca. 17ms … 30ms) antwortet das Modem jeweils mit einem 8 Byte langen Datenrahmen, der Statusinformationen und Daten des angesprochenen Moduls enthält. Das Betriebssystem der C-Control II Unit enthält Routinen zur seriell-synchronen Kommunikation mit dem 2W-Bus-Modem. Ihr C2-Programm muß lediglich die Datenrahmen aufbereiten und auswerten. Weitere Hinweise zur Programmierung finden Sie im Kapitel 7.12.
Jedes 2W-Bus-Modul hat eine eindeutige 8bit-Adresse im Netzwerk. Diese Adresse muß einmalig in das Modul übertragen werden. Ein Beispielprogramm zur Einstellung der 2W-Bus-Moduladressen finden Sie auf der CD zur C-Control II Unit.

3.3.8.4 Druckerschnittstelle

An der C-Control II Unit kann ein Drucker angeschlossen werden, wenn sich dieser zeichenweise im klassischen ASCII-Modus ansteuern läßt. Problemlos sollten sich Nadeldrucker verwenden lassen. Tintenstrahl- und Laserdrucker arbeiten in der Regel seitenweise. Die Ausgabe einer Seite erfolgt dann oft erst nach Empfang des Seitenvorschubzeichens. Einige neue Drucker sind auf die Zusammenarbeit mit grafischen Betriebssystemen spezialisiert (sogenannte “GUI-Printer”) und benötigen zur Ansteuerung besondere Treiber. Diese Drucker eignen sich nur bedingt zur Verwendung mit der C-Control II Unit (den Druckertreiber für C-Control II müßten Sie selbst implementieren). Bitte haben Sie Verständnis, daß es Conrad Electronic aufgrund der Vielzahl der existierenden Druckermodelle nicht möglich ist, in jedem Fall eine Aussage zu treffen, welcher Drucker direkt an der C-Control II Unit betrieben werden kann und welcher nicht.
Zum Anschluß eines Druckers an die C-Control II Unit verbinden Sie folgende Pins mit denen einer 25poligen SUB-D-Buchse:

Signal	C-Control II Pin	SUB-D 25 Pin
Datenbit 0	P1L.0	2
Datenbit 1	P1L.1	3
Datenbit 2	P1L.2	4
Datenbit 3	P1L.3	5
Datenbit 4	P1L.4	6
Datenbit 5	P1L.5	7
Datenbit 6	P1L.6	8
Datenbit 7	P1L.7	9
BUSY (Handshake)	P1H.5	11
ACKNOWLEDGE (Handshake)	P1H.6	10
STROBE (Handshake)	P1H.7	1
Masse	GND	18 … 25

Das Betriebssystem verfügt über eingebaute Routinen zur Datenausgabe an den Drucker über die genannten Ports. Bei Bedarf können Sie die erweiterten Signale des Druckers auswerten bzw. an den Drucker senden. Diese Ansteuerung ist jedoch nicht im Betriebssystem der C-Control II Unit implementiert. Welche erweiterten Signale die Schnittstelle eines Druckers definiert, entnehmen Sie bitte der Fachliteratur. Mit welchen freien Digitalports der C-Control II Unit sie die zusätzlichen Signale verbinden, steht Ihnen frei.
Hinweise zum Zugriff auf die Druckerschnittstelle in Ihren Programmen finden Sie im Kapitel 7.5.

3.3.9 A/D-Ports

Die C-Control II Unit verfügt über 8 Ports (A/D 0 … A/D 7), die mit dem internen 10bit-A/D-Wandler des Mikrocontrollers verbunden sind. Das Betriebssystem nimmt im Hintergrund ständig A/D-Wandlungen vor. Zur Reduzierung von Störeinflüssen werden die Spannungssignale durch eine gleitenden Mittelwertbildung gefiltert. Lesen Sie zum Thema A/D-Wandlung auch die Kapitel 3.2.3 und 7.9.7.

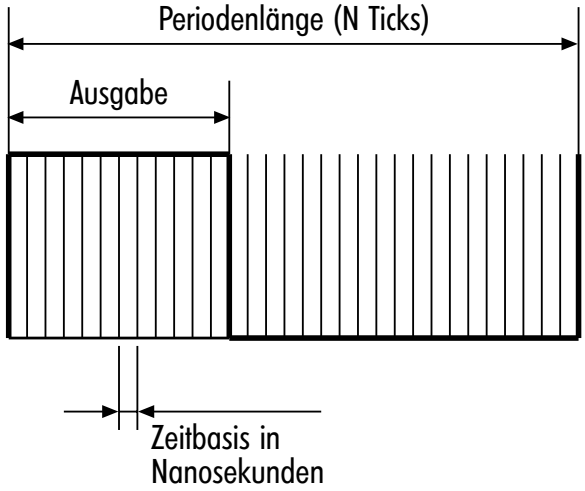
3.3.10 DCF/FRQ-Ports

Am DCF/FRQ0 -Pin kann der invertierte Signalausgang einer DCF77-Funkuhraktivantenne angeschlossen werden. Der erforderliche Pullup-Widerstand ist bereits in der Unit integriert. Das Betriebssystem übernimmt bei Signalempfang automatisch die Dekodierung der Datenrahmen und stellt die interne Uhr des Systems.
Der DCF-Pin kann gleichzeitig zur Messung von Pulsfrequenzen von 100Hz bis ca. 30kHz

benutzt werden, ebenso der zweite Frequenzmeßpin FRQ1. Die Frequenzmessung erfolgt nach dem Prinzip der Pulszählung in einer Torzeit von einer Sekunde. Dadurch ergeben die Zählwerte direkt eine Frequenz in Hz.
Weitere Hinweise zu Funkuhrempfang und Frequenzmessung finden Sie in den Kapiteln 7.11 und 7.9.6.

3.3.11 PLM-Ports

Die C-Control II Unit verfügt über drei Ports ("Kanäle") zur Ausgabe pulslängenmodulierter Signale: PLM0, PLM1 und BEEP/PLM2. Diese können zur D/A-Wandlung, zur Ansteuerung von Servomotoren im Modellbau oder zur Ausgabe von Tonfrequenzen benutzt werden. Ein pulslängenmoduliertes Signal hat eine Periode von N sogenannten "Ticks". Die Dauer eines Ticks ist die Zeitbasis. Setzt man den Ausgabewert eines PLM-Ports auf X, dann hält dieser für X Ticks einer Periode Highpegel und fällt für den Rest der Periode auf low.
Als Zeitbasen können 8 verschiedene Zeiten von 400ns bis 51.2µs eingestellt werden. Die Periodenlänge kann von 0 bis 65535 variiert werden.



Die PLM-Kanäle 0 und 1 haben eine gemeinsame Zeitbasis und Periodenlänge. Für Kanal 2 kann eine von den Kanälen 0 und 1 unabhängige Zeitbasis und Periodenlänge eingestellt werden. Lesen Sie dazu auch Kapitel 7.8.
In Anwendungen zur pulsweitenmodulierten Digital-Analogwandlung werden Zeitbasis und Periodenlänge einmalig eingestellt und dann nur der Ausgabewert manipuliert. Hält man jedoch die Periodenlänge variabel und stellt sicher, daß der Ausgabewert z.B. stets die Hälfte der Periodenlänge beträgt, können die PLM-Kanäle auch zur Ausgabe von Rechtecksignalen bestimmter Frequenzen benutzt werden. Die Ausgabefrequenz eines

PLM-Kanals ergibt sich aus 1 / (Zeitbasis * Periodenlänge). Hinweise zur Ausgabe von Tonfrequenzen finden Sie im Kapitel 7.8.5
Die PLM-Ports sind nach ihren elektrischen Eigenschaften Digitalports. Beachten Sie die technischen Randbedingungen für Digitalports (max. Strom).

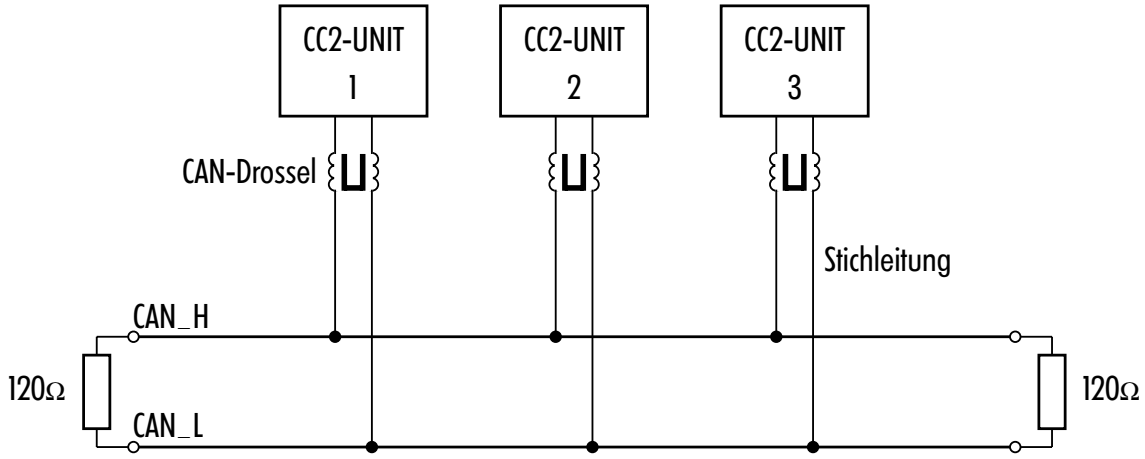
3.3.12 CAN-Interface

CAN - "Controller Area Network" - ist ein digitales Kommunikationssystem zur Vernetzung von mikrocontrollerbasierten Baugruppen. Ursprünglich wurde es für Anwendungen in der Automobiltechnik konzipiert und geht auf Entwicklungen der Robert Bosch GmbH zurück. In modernen Autos übernehmen Mikrocontroller mit CAN Steuerungsaufgaben von ABS bis zur Zentralverriegelung. Inzwischen hat sich CAN auch als Feldbus in der industriellen Automatisierungstechnik etabliert und verbindet speicherprogrammierbare Steuerungen mit Sensoren und Aktoren oder auch untereinander.
Der CAN-Bus selbst ist ein Kabelstrang mit zwei elektrischen Leitern (CAN-H, CAN-L). Die maximale Länge des CAN-Bus beträgt bis zu 1200m, abhängig von der genutzten Bitrate.

Bitrate	maximale Buslänge
50 kbit/s	1200 m
62,5 kbit/s	1000 m
125 kbit/s	500 m
250 kbit/s	250 m
500 kbit/s	100 m

Auf dem CAN-Bus werden Datenbits massefrei als Differenzspannungen übertragen (vgl. RS485 Standard). Diese Art der Datenübertragung macht das Signal sicherer gegen die Einstrahlung von Störsignalen, verglichen mit der Übertragung von Spannungspegeln mit Bezugsmasse. An beiden Enden muß der CAN-Bus mit je einem 120 Ohm Widerstand abgeschlossen sein. Die Längen der Stichleitungen vom Bus zu den angeschlossenen Baugruppen sollten einige wenige Zentimeter nicht überschreiten. Sind die Stichleitungen zu lang oder fehlen die Abschlußwiderstände, können Leitungsreflexion die Datenübertragung stark stören oder unmöglich machen. Als Kabelstrang können einfache, verdrillte Leitungen eingesetzt werden ("twisted pair"). Bei der Verwendung geschirmter Leitungen, lassen sich Probleme mit Störungsabstrahlung und -einstrahlung reduzieren. Das gilt besonders für lange CAN-Busse und hohe Übertragungsgeschwindigkeiten. Zur Unterdrückung von Gleichtaktstörungen sollten spezielle CAN-Drosseln verwendet werden. Die CAN-Übertragungsgeschwindigkeit von 1Mbit/s wird von C-Control II nicht direkt

unterstützt. Diese Bitrate erfordert besondere Maßnahmen zur Übertragungssicherheit und vor allem gegen die Störaussendung. Conrad Electronic geht nicht davon aus, daß die C-Control II Unit in Applikationen Einsatz findet, die derartige hohe Übertragungsraten erforderlich machen.
Folgende Zeichnung zeigt die Vernetzung von 3 C-Control II Units über den CAN-Bus:



Der Anschluß einer C-Control II Unit erfolgt über die Pins CANH und CANL. Die digitalen Signale an den Pins CAN_Tx und CAN_Rx dürfen nur zu Diagnosezwecken benutzt werden!
Einen Überblick über die verfügbaren Funktionen zur Programmierung einer der CAN-Applikation bekommen Sie in Kapitel 7.1.

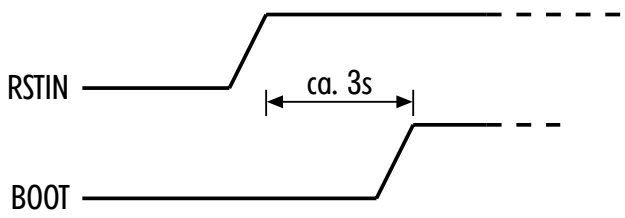
4 Betriebssystem

4.1 Überblick

Das Betriebssystem der C-Control II Unit übernimmt das gesamte Interrupt-Handling, die Konfiguration des Mikrocontrollers nach dem Reset, das Laden von Anwenderprogrammen in den FLASH-Speicher sowie den Start und die Ausführung von Anwenderprogrammen. Während des Programmablaufes steuert das System alle Ein- und Ausgabeoperationen im Hintergrund der Anwendung. Ein wichtiger Teil des Systems ist die virtuelle Maschine (siehe unten) zur Ausführung von C2-Programmen.
Das Betriebssystem wurde in Assembler und der Programmiersprache C geschrieben und liegt in Binärform auf der CD zur Unit vor. Das Betriebssystem der C-Control II Unit ist bei der Auslieferung in der Regel noch nicht installiert und muß erst im ersten Segment des FLASH-Speichers gespeichert werden (erstes Segment = "Segment 0"). Nur in Ausnahmefällen finden Sie dort eine Testversion vor, die während der Produktion geladen wurde. Wenn Sie die C-Control II Unit erstmalig in Betrieb nehmen, sollten Sie auf jeden Fall zunächst das Betriebssystem von der CD installieren. Auf der C-Control Homepage www.c-control.de im Internet finden Sie gegebenenfalls auch eine aktuellere Version der Installation oder einzelne Dateien zum Download. Sie sollten dann diese Version statt der auf der CD ausgelieferten verwenden.

4.2 Bootstrap - Installieren des Betriebssystems

Im Bootstrap-Modus des Mikrocontrollers kann das C-Control II -Betriebssystem in den FLASH-Speicher der Unit übertragen werden. Installieren Sie zunächst das Boot-Tool von der C-Control-CD auf Ihrem PC. Lesen Sie dabei die Installationsanleitung und gegebenenfalls die zusätzlichen Hinweise.
Um den Bootstrap-Modus zu aktivieren, müssen Sie bei einem Hardware-Reset der C-Control II Unit gleichzeitig den BOOT-Pin auf Low-Pegel legen, z.B. durch Kurzschließen des BOOT-Pins mit dem benachbarten GND-Pin durch eine Steckbrücke oder einen Taster. Wenn kurz darauf der BOOT-Pin wieder auf High-Pegel geht, erwartet der Mikrocontroller die Übertragung des Betriebssystems über die serielle Schnittstelle.

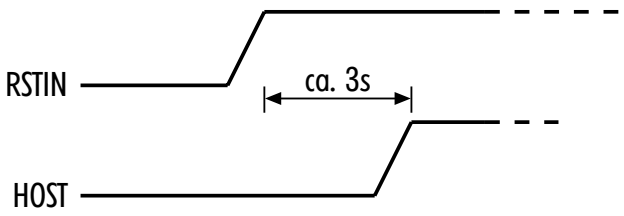


Starten Sie nun die Datenübertragung des Systems mit Hilfe des Boot-Tools. Dieses PC-Programm übernimmt die gesamte Steuerung des Boot-Vorganges: Die Übertragung beginnt mit einem Nullbyte (1 Startbit, 8 Datenbits = 0, 1 Stopbit) Der Mikrocontroller der C-Control II Unit empfängt das Nullbyte und benutzt es zur Messung der Übertragungsgeschwindigkeit (z.B. 9600 Baud). Dann initialisiert er seine eigene Schnittstelle entsprechend. Als Antwort sendet der Controller ein Identifizierungsbyte an den PC. Die PC-Software erkennt den konkreten Controller-Typ und überträgt einen Ur-Loader (32 Bytes) an die C-Control II Unit. Dieser Ur-Loader wird vom Controller im internen RAM abgelegt und automatisch gestartet. Der Ur-Loader selbst ist ein minimales Programm, das nichts weiter tut, als die zweite Stufe des Loaders über die serielle Schnittstelle zu empfangen, im RAM abzulegen und anschließend zu starten. Die zweite Stufe des Loaders empfängt letztendlich das Betriebssystem und speichert es im ersten Segment des externen FLASH. Dieses erste Segment enthält ab Adresse 0x0000 die Interrupt-Vektoren, darunter auch den Reset-Vektor. Wenn das Betriebssystem korrekt installiert wurde, startet es nach dem nächsten Reset automatisch und geht in den Hostmodus über.
Bei allen nachfolgenden Kapiteln gehen wir davon aus, daß Sie das Betriebssystem der C-Control II Unit korrekt installiert haben.

4.3 Hostmodus

4.3.1 Systeminitialisierung und automatisches Starten von Anwenderprogrammen

Nach dem Reset werden die Ressourcen des Mikrocontrollers initialisiert. Anschließend wird geprüft, ob ein C2-Anwenderprogramm im FLASH gespeichert ist. Wird ein Programm gefunden, dann startet das System die virtuelle Maschine und führt dieses Programm aus. Anderenfalls geht das System in den Hostmodus über.
Wenn Sie ein im FLASH gespeichertes Anwenderprogramm nicht automatisch starten wollen, z.B. weil Sie ein neues Anwenderprogramm in die Unit laden möchten, muß beim Reset der HOST-Pin der Unit auf Low-Pegel gezogen sein.



4.3.2 Download von Anwenderprogrammen und andere Host-Befehle

Im Hostmodus erwartet das System den Empfang von Kommandobytes über die serielle Schnittstelle, die es dann ausführt. Der wichtigste Befehl ist der zum Start des Programm-Download (CMD_LOAD_VMC). Beim Download wird ein kompiliertes Anwenderprogramm (VMC-Datei) vom PC zur C-Control II Unit übertragen und von dieser im FLASH-Speicher abgelegt. Die Übertragung erfolgt innerhalb der Integrierten Entwicklungsumgebung, die Sie von der Utility-CD installieren können. Lesen Sie dazu die Hinweise auf der CD, bzw. in der Online-Hilfe zur Integrierten Entwicklungsumgebung.
U.a. sind folgende Kommandobytes definiert:

Kommando	Reaktion der Unit im Hostmodus
CMD_SEND_ID (0)	antwortet mit dem Text "C-Control II"
CMD_SEND_VERSION (2)	antwortet mit einem Versionsstring
CMD_START (3)	startet ein vorhandenes C2-Anwenderprogramm
CMD_LOAD_VMC (4)	Beginn der Übertragung eines C2-Anwenderprogramms
CMD_ERASE_VMC (6)	löscht ein vorhandenes C2-Anwenderprogramm
CMD_RESET (255)	führt einen Software-Reset des Mikrocontrollers aus

4.4 Virtuelle Maschine

4.4.1 Grundlagen

Die Ausführung von Anwenderprogrammen auf der C-Control II Unit erfolgt durch die virtuelle Maschine (VM).
Der Begriff "virtuell" wird heute in vielen Zusammenhängen verwendet. "Virtuell" bedeutet soviel wie "gedacht", "nachgebildet", "so als ob"… . Die "Maschinen" der Computertechnik sind die Mikroprozessorkerne. Sie sind durch ihren jeweiligen Aufbau aus Registern, Rechenwerken, Steuerwerken und der Schnittstelle zum Programm- und Datenspeicher charakterisiert. Diesen Aufbau bezeichnet man als die Prozessorarchitektur.
Eine virtuelle Maschine ist ein nachgebildeter Mikroprozessorkern. Die gewünschte

Architektur wird per Software auf einem existierenden Computersystem hergestellt. Dem Nachteil einer geringeren Geschwindigkeit bei der Ausführung einfacher Programmanweisungen stehen einige Vorteile gegenüber:

- Realisierbarkeit von Architekturelementen, die das als Basis benutzte Computersystem nicht bietet
- optimale Anpassung an eine Programmiersprache
- einfache Portierbarkeit der VM auf andere Computersysteme bei Wiederverwendbarkeit existierender Programme, sogar als Binärcode

Im Betriebssystem der C-Control II Unit läuft eine von Conrad Electronic entwickelte virtuelle Maschine. Sie ist funktionell eng an die Programmiersprache C2 gekoppelt und bietet den Anwenderprogrammen einfachen Zugang zu den Systemressourcen.

4.4.2 Binärcodeinterpreter

Der C2-Compiler erzeugt aus dem Programmquelltext des Anwenderprogramms einen Binärcode. Dieser kann in die C-Control II Unit geladen werden. Bei der Ausführung des Anwenderprogramms wird der Binärcode schrittweise gelesen und interpretiert. D.h. für jeden gelesenen Code wird eine definierte Operation ausgeführt.
Die virtuelle Maschine der C-Control II Unit kennt vier Codeformen - zwei 16bit-Formen und zwei 32bit-Formen:

Form	low-byte (low-word)	high-byte (low-word)	high-word
A	Befehl (0 … 63)	-	-
B	Befehl (64 … 127)	Byte-Parameter	-
C	Befehl (128 … 191)	-	Word-Parameter
D	Befehl (192 … 255)	Byte-Parameter	Word-Parameter

An dem Wertebereich, in dem der Befehlscode liegt, erkennt der Interpreter die Befehlsform. Abhängig von der Befehlsform werden optional ein Byte-Parameter und ein Word-Parameter geladen, bevor es zu Ausführung der dem Codewert zugeordneten Operation kommt.
Einen Überblick über alle verfügbaren Operationen zu geben, führt an dieser Stelle zu weit und ist für das prinzipielle Verständnis der C-Control II Unit nicht erforderlich.

4.4.3 Multithreading

Ein Hauptmerkmal der C-Control II Unit ist die Unterstützung von Multithreading in Anwenderprogrammen. Verschiedene Programmteile können quasi gleichzeitig und voneinander unabhängig abgearbeitet werden. Dadurch lassen sich komplexe, in der Applikation parallel und asynchron ablaufende Vorgänge auf einfache Weise behandeln.

Beispiel:
verschiedene Digitalports sollen ständig überwacht werden; bei Eintreten einer bestimmten Kombination ist eine Pulsfolge mit vorgegebenem Timing auszugeben; gleichzeitig sind ständig einige A/D-Kanäle zu überwachen und bei Überschreitung von Grenzwerten soll ein Alarm ausgelöst werden; von der seriellen Schnittstelle werden in einem bestimmten Rhythmus Datenrahmen erwartet, die ausgewertet und beantwortet werden sollen; über den Drucker sollen Meßwerte ausgedruckt werden …
In Programmteilen, die mit anderen Geräten kommunizieren, kann es Situationen geben, in denen gewartet werden muß, bis der Kommunikationspartner bereit für den Datenempfang ist. In einem Computersystem mit ausschließlich sequentieller Abarbeitung des Programmes ist es praktisch unmöglich, in diesen Wartezuständen auf weitere Ereignisse zu reagieren. So kann es passieren, daß ein Alarmzustand aufgrund einer Übertemperatur nicht oder nicht rechtzeitig erkannt wird, während das System auf die Bereitschaft eines angeschlossenen Druckers wartet.
Beim Multithreading der C-Control II Unit kann ein Programm in bis zu 255 Threads ("Fäden") aufgetrennt werden. Jedem Thread wird vom Kern des Betriebssystems reihum eine Portion Rechenzeit zugeteilt. Wieviel Rechenzeit ein Thread erhält, kann über seine Priorität gesteuert werden. Bei Priorität 0 wird sofort zum nächsten Thread weitergeschaltet. Der höchstmögliche Prioritätswert ist 255. In einem Umlauf führt der Binärcodeinterpreter für jeden Thread maximal so viele Operationen aus, wie es dessen jeweiligem Prioritätswert entspricht. In Wartesituationen erfolgt die Weiterschaltung vorzeitig. Die Priorität jedes Threads kann während des Programmablaufes den aktuellen Leistungsanforderungen angepaßt werden.
Die Schwierigkeit bei der Erstellung eines Programms mit Multithreading liegt in der ausgewogenen Vergabe der Prioritätswerte. Bei bis zu 255 Threads und 256 Prioritätsstufen gibt es nahezu unzählige Möglichkeiten, Rechenzeit zu verteilen. Mit der Zuteilung von sehr niedrigen Prioritäten an alle Threads ergeben sich schnelle Umlaufzeiten und somit eine relativ geringe Verzögerung, bis ein einzelner Thread auf ein Ereignis reagieren kann. Dafür sinkt die Performance des Gesamtsystems, da pro Zeiteinheit mehr Rechenleistung der virtuellen Maschine für das Umschalten von Threads verbraucht wird. So ist es nicht

sinnvoll, allen Threads die Priorität 1 zu erteilen. Die Performance steigt mit der Vergabe von hohen Prioritäten. Das wird jedoch durch längere Reaktionszeiten erkauft. Im nicht zu empfehlenden Extremfall haben alle Threads die Priorität 255.
Es hat sich bewährt, die meisten Threads mit einer eher niedrigen Standardpriorität (z.B. 32) laufen zu lassen. Nur einigen Programmabschnitten, die lange Zeit auf ein Ereignis warten, dann aber mit hoher Geschwindigkeit reagieren müssen, sollte ein höherer Wert zugeteilt werden.

Anmerkung - Multithreading vs. Multitasking:
Von Tasks spricht man in der Regel im Zusammenhang mit parallel unter einem Betriebssystem laufenden, unabhängigen Programmen, z.B. einer Textverarbeitung, einem E-Mail-Client und einer Datenbank, die gleichzeitig auf einem PC gestartet wurden. Threads hingegen sind sogenannte "leichtgewichtige Prozesse" innerhalb eines Programms. Mehrere Threads eines Programms teilen sich einen gemeinsamen Adreßraum und können über globale Variablen relativ einfach Daten austauschen. Die Frage, ob die C-Control II Unit Multithreading oder Multitasking betreibt, wenn sie eine Leuchtdiode blinken läßt und parallel Daten von der seriellen Schnittstelle empfängt, ist eher von akademischer als von praktischer Bedeutung. Für den Anwender genügt zu wissen, daß es funktioniert.

4.4.4 Programm- und Konstantenspeicher

Die maximale Länge des Binärcodes beträgt 128kB. Er findet in zwei Segmenten des FLASH-ROMs Platz. Die Adressierung eines Binärcodes im Programmspeicher erfolgt word-weise über einen 16bit-Offset.
Getrennt vom Programmspeicher nutzt die virtuelle Maschine zwei weitere FLASH-Segmente für 128kB Konstantenspeicher. Am Anfang des Konstantenspeichers sind die Initialisierungswerte für jeden Thread des Anwenderprogramms abgelegt. Dann folgen konstante Zahlenwerte, Tabellen und Strings, die im Programm verwendet werden. Der Zugriff auf den Konstantenspeicher durch das Anwenderprogramm erfolgt word-weise durch spezielle Binärcodes und einen 16bit-Offset.

4.4.5 Datenspeicher

Der dritte Speicherblock ist der Datenspeicher, der sich im externen SRAM der C-Control II Unit befindet. Von den 64kB stehen ca. 60kB für Daten des Anwenderprogramms zur Verfügung, abzüglich des Speicherbedarfes für den Stapelprozessor. Die Adressierung der Daten erfolgt byte-weise über einen 16bit-Offset.

4.4.6 Stapelprozessor

Die virtuelle Maschine der C-Control II Unit arbeitet als Stapelprozessor. Sie implementiert nicht wie viele Mikrocontroller und Mikroprozessoren spezielle Rechenregister oder einen Akkumulator. Statt dessen werden alle Operanden auf einen Stapel (Stack) geladen. Die Operationen des virtuellen Prozessors manipulieren stets den obersten Wert auf dem Stapel oder verknüpfen die zwei obersten Werte zu einem Ergebnis. Speicheroperationen nehmen einen Wert vom Stapel und legen ihn an einer Adresse im Datenspeicher ab.
Der Stapelprozessor der C-Control II Unit unterstützt Rechenoperationen mit vorzeichenbehafteten 16bit- und 32bit-Integerwerten sowie mit 64bit-Fließkommazahlen. Bytes werden immer als 16bit-Integer verarbeitet.
Der Stapel dient auch als Zwischenspeicher für lokale Variablen von Threads und Unterfunktionen sowie zur Übergabe von Parametern und Rückgabewerten beim Aufruf von Unterfunktionen. Außerdem werden Rücksprungadresse und Speicherkontext (BP) vor einem Funktionsaufruf auf dem Stapel gesichert und beim Rücksprung wiederhergestellt.
Jeder Thread des Anwenderprogramms verfügt über einen eigenen Stapel. Der für den Stapel eines Threads zur Verfügung stehende Speicherplatz beträgt theoretisch 64kB. Eine Limitierung ist jedoch dadurch gegeben, daß sich der Stapel das 64kB große SRAM-Segment mit einigen Daten des Betriebssystems, den globalen Variablen des Anwenderprogramms und weiteren Stapeln anderer Threads teilen muß.
Die Adressierung von Daten auf dem Stapel erfolgt relativ zu einem 16bit-Basepointer (BP). Ein 16bit-Stackpointer (SP) zeigt auf das obere Ende des Stapels. Jeder Thread hat sein eigenes Paar von Base- und Stackpointern.

4.4.7 Systemschnittstelle

Die virtuelle Maschine der C-Control II Unit verfügt über spezielle Befehlscodes als Schnittstelle zu den Hardwareressourcen und Funktionen des Betriebssystems. Damit unterscheidet sie sich vom zugrundeliegenden Mikrocontroller C164CI, der die Hardwareressourcen als Register in einen bestimmten Speicheradreßbereich legt (Special Function Register - "SFR"). Die Umwandlung von Systemoperationen der virtuellen Maschine in konkrete Hardwarezugriffe, z.B. auf Register in den SFR, erfolgt im Betriebssystem der C-Control II Unit. Damit sind die virtuelle Maschine selbst sowie die dafür compilierten Anwenderprogramme relativ einfach auf andere Computersysteme portierbar.
Die Befehlscodes zum Zugriff auf Hardwareressourcen werden den Anwenderprogrammen über inline-Funktionen in den C2-Systemmodulen zur Verfügung gestellt.

5 Die Programmiersprache C2

5.1 Einleitung

Die Programmierung der C-Control II Unit erfolgt in der Programmiersprache C2. C2 ist syntaktisch ähnlich zu C, einige Details erinnern auch an PASCAL oder BASIC. Wie in C gibt es nur eine überschaubare Anzahl von Schlüsselworten. Einige Schlüsselworte dienen speziell der Unterstützung des Multithreading. Alle System- und Spezialfunktionen werden über Bibliotheksmodule zur Verfügung gestellt und können in Projekte eingebunden werden. Die Projektarbeit in C2, in Verbindung mit der Integrierten Entwicklungsumgebung, ist wesentlich einfacher als in C. C2 bietet alle Möglichkeiten, die zur strukturierten Programmierung benötigt werden. Selten verwendete, besonders "gefährliche" und schwer verständliche Sprachkonstrukte von C wurden weggelassen.
Inhalt dieses Kapitels ist die systematische Beschreibung der Programmiersprache C2 in Form einer Referenz. Nach einem Überblick über die Syntaxelemente folgt eine ausführliche Darstellung aller Operatoren, Typen, Definitions- und Anweisungsformen. Abschließend finden Sie eine Aufstellung aller Datentypen und Funktionen der Systemmodule sowie kurze Beispiele zu deren Anwendung.
Ausführlichere Programmbeispiele befinden sich auf der CD zur Integrierten Entwicklungsumgebung.
Für nachfolgende Abschnitte vereinbaren wir die folgenden Formatierungen und Stile, um Textelemente mit besonderer Bedeutung gezielt hervorzuheben.

datei.ext	Dateinamen
[STRG]+[F1]	Tasten und Tastenkombinationen
`sourcecode`	Quelltextbeispiele
`Name`	im Quelltext zu ersetzen durch das beschriebene Syntaxelement

5.2 Projekte und Module

Der C2-Compiler erzeugt aus einem C2-Projekt einen Binärcode, der anschließend in die C-Control II Unit übertragen und von der virtuellen Maschine als Programm ausgeführt werden kann.

Ein C2-Projekt kann aus beliebig vielen Modulen bestehen. Ein Modul ist eine einfache ASCII-Textdatei mit der Dateierweiterung “c2”. Der Dateiname - ohne Pfad und Extension - ist der Modulname. Der Name eines Moduls muß ein gültiger C2-Bezeichner sein (siehe unten). Jeder Name darf nur einmal im Projekt vorkommen.
Auf die Module verteilt steht der gesamte Quelltext eines Programms. Auf Modulebene werden globale Variablen, benannte Konstanten, zusammengesetzte Datentypen, Funktionen und Threads definiert.
Die Aufteilung eines Projektes in mehrere Module hat folgende Vorteile

- verbesserte Übersichtlichkeit über die Quelltexte großer Programme
- einfache Wiederverwendung getesteter Module in anderen Projekten
- verbesserte Lesbarkeit von Quelltexten durch die automatische Bildung eines Namensraumes für jedes Modul und die Notwendigkeit der Modulspezifikation bei der Verwendung von Bezeichnern aus einem Modul

Ein Projekt beschreibt eine Liste von Modulen. Die Reihenfolge der Module in der Liste bestimmt die Reihenfolge bei der Übersetzung durch den C2-Compiler. Das erste Modul in der Liste wird als erstes übersetzt, dann das zweite usw. bis zum letzten Modul.
Zusammen mit der Integrierten Entwicklungsumgebung werden zahlreiche Bibliotheksmodule ausgeliefert, die z.B. Funktionen zum Zugriff auf Systemressourcen der C-Control II Unit enthalten. In der Praxis besteht ein Projekt zunächst aus einigen dieser Bibliotheksmodule. Dann folgen in der Liste die wiederverwendbaren Anwendermodule, z.B. zur Implementierung oft benötigter Algorithmen, wie standardisierte Prüfsummenberechnungen oder ähnliches. Anschließend stehen Module mit applikationsspezifischem Code, z.B. zur Ansteuerung einer ganz konkreten externen Hardware. Die letzten Module der Liste sind in der Regel die Hauptmodule. Sie enthalten unabhängige Hauptthreads des Programms. Die meisten Anwendungen haben nur einen Hauptthread und somit nur ein Hauptmodul am Schluß der Modulliste.

5.3 Syntax - Grundelemente

5.3.1 Kommentare

Sinnvolle Kommentare in einem Programm können dessen Verständlichkeit und Lesbarkeit erhöhen. C2-Kommentare sind kompatibel zu denen in C und C++. Es gibt Zeilenendkommentare, die durch zwei unmittelbar aufeinanderfolgende Schrägstriche //

eingeleitet werden. Jeglicher Text bis zum Zeilenende, einschließlich der Schrägstriche, wird beim Compilieren überlesen.
z.B.:

```
a = 123;  // das ist ein Kommentar
```

Mehrzeilige Kommentare können in **/* */** -Sequenzen eingebettet werden.
z.B.:

```
/*
    das alles
      ist ein
         Kommentar
*/
```

Verschachtelte mehrzeilige Kommentare sind nicht zulässig.

5.3.2 Zwischenräume

Alle Zeichen mit den ASCII-Codes 0 … 32 werden als Zwischenräume (engl. "whitespaces" oder "blanks") gelesen und beim Compilieren überlesen, z.B. die Leerzeichen, Tabulatoren und Zeilenvorschübe im Programmquelltext.
Das gilt jedoch nicht innerhalb von Stringkonstanten. Leerzeichen in Stringkonstanten bleiben erhalten und gelangen so zur Ausgabe, wie sie sich im Quelltext befinden.

5.3.3 Bezeichner

Bezeichner sind die Namen von Modulen, Variablen, Konstanten, zusammengesetzten Typen, deren Felder, Funktionen und Threads.

- ein Bezeichner besteht aus mindestens einem Zeichen und kann beliebig lang sein
- gültige Zeichen eines Bezeichners sind Buchstaben (A … Z, a … z, keine Umlaute oder ß), Ziffern (0 … 9) und Unterstriche (_)
- das erste Zeichen darf keine Ziffer sein
- C2 ist case-sensitiv, d.h. Groß- und Kleinschreibung von Buchstaben werden unterschieden - Abc, abc, aBc … sind verschiedene Bezeichner
- C2-Schlüsselworte sind als Bezeichner nicht zulässig

☞ Jeder Bezeichner muß dem Compiler vor seiner ersten Verwendung bekannt sein.

D.h. er muß weiter oben im aktuellen Modulquelltext oder in einem vorher übersetzten Modul definiert sein. Bezeichnern, die in einem anderen Modul zuvor definiert sind, muß

ohne Zwischenraum der Modulname und ein Punkt vorangestellt werden.
Beispiel:
Funktion `fx`, definiert in Moduldatei a.c2

```
function fx ()
{
  // ...
}
```

Aufruf der Funktion weiter unten in a.c2

```
fx();
```

Aufruf der Funktion in einem anderen Modul (in der Modulliste des Projektes nach a.c2)

```
a.fx();
```

Auf alle Bezeichner von globalen Variablen, benannten Konstanten, zusammengesetzten Datentypen, Funktionen und Threads eines Modules kann in nachfolgenden Modulen auf die hier beschriebene Weise zugegriffen werden.

5.3.4 Anweisungen und Anweisungsblöcke

Anweisungen sind die Grundbausteine eines Computerprogramms. Folgende Anweisungsformen werden in C2 unterschieden:

- Variablendefinition
- Konstantendefinition
- Zuweisung
- Funktionsaufruf
- Programmsteueranweisung

Eine Anweisung kann sich über eine oder auch über mehrere Zeilen erstrecken.

☞ Nach jeder Anweisung muß ein Semikolon stehen.

z.B.:

```
int a;
a = 123;
```

Anweisungsblöcke sind Folgen von Anweisungen, die durch geschweifte Klammern **{ }** zusammengefaßt sind.

z.B.:

```
{
  a = 123;
  b = a + 1;
}
```

Nach einem Anweisungsblock ist kein Semikolon erforderlich. Anweisungsblöcke können statt einer einzelnen Anweisung stehen, z.B. um mehrere Aktionen innerhalb einer Programmsteueranweisung auszuführen.

z.B.:

```
if x > 0
{
  a = 123;
  b = a + 1;
}
```

5.3.5 Ausdrücke

Ein Ausdruck (oder "Term") ist die Verknüpfung von Daten (Variablen oder Konstanten) durch Operatoren. In C2 gibt es ausschließlich numerische Ausdrücke. Jeder numerische Ausdruck ergibt durch mathematische Berechnung einen Wert.
Gültige numerische Ausdrücke sind z.B.

```
a + b * c
1 + x
f(x) + c
1 + 2 + 1977
```

Eine Sonderform ist der konstante Ausdruck. Dessen Wert läßt sich bereits vor der Programmausführung bestimmen. So ist der Wert des Ausdrucks 1 + 2 + 1977 offenbar immer 1980. Das steht zur Zeit der Programmierung fest und wird sich auch bei der Programmausführung nicht ändern. Um unnötige Berechnungen konstanter Ausdrücke während der Programmausführung zu vermeiden, versucht der C2-Compiler, diese

weitestgehend vorherzubestimmen und zusammenzufassen. So wird die Anweisung

```
a = 1 + 2 + 1977 + c
```

vom Compiler vorberechnet und intern umgewandelt in

```
a = 1980 + c
```

Funktionen (siehe 5.8) werden jedoch immer aufgerufen und ausgeführt, auch wenn deren Parameter und der Rückgabewert konstant sind.
In konstanten Ausdrücken sind auch zuvor definierte benannte Konstanten (siehe 5.6) verwendbar.

5.3.6 Schlüsselworte

Untenstehend finden Sie eine alphabetische Liste aller C2-Schlüsselworte. Detaillierte Definitionen und Anwendungsbeispiele finden Sie im weiteren Verlauf dieser Anleitung.

and	**function**	**or**	**step**
break	**halt**	**quit**	**string**
byte	**if**	**release**	**thread**
capture	**inline**	**resume**	**type**
const	**int**	**return**	**wait**
continue	**long**	**returns**	**while**
do	**loop**	**run**	**xor**
else	**nand**	**shl**	**yield**
float	**nor**	**shr**	
for	**not**	**sleep**	

5.4 Datentypen

5.4.1 Numerische Datentypen

C2 bietet insgesamt vier verschiedene numerische Datentypen zur Definition von Variablen, Funktionsparametern und Rückgabewerten von Funktionen: **byte, int, long** und **float**. Der Datentyp einer Variablen, eines Funktionsparameters oder eines Rückgabewertes sollte nach dem erforderlichen Wertebereich und der notwendigen

Rechengenauigkeit gewählt werden. Operationen mit `long-` und `float-`Daten führen zu einem wesentlich höherem Bedarf an Speicherplatz und Rechenzeit. Die Ausführungsgeschwindigkeit von `float-`Operationen ist geringer als die von `long-`Operationen. Diese wiederum dauern etwas länger als `int-`Berechnungen. Das Rechnen mit Bytes anstelle von Integerdaten bringt keinen Geschwindigkeitsvorteil, da Bytes vom Stapelprozessor der virtuellen Maschine immer zu Integern erweitert werden. Das Verwenden des **`byte`** -Typs bei der Definition globaler und lokaler Variablen spart jedoch etwas Speicherplatz.

Typschlüsselwort	Wertbereich	Speicherplatzbedarf für Variablen
`byte`	0 … 255	1 Byte
`int`	-32768 … 32767	2 Bytes
`long`	- 2147483648 … 2147483647	4 Bytes
`float`	$\pm 1.7*10^{-308}$ … $\pm 1.7*10^{308}$	8 Bytes

5.4.2 Zeichenketten (Strings)

Die häufigste Zeichenkettenoperation in Steuerungssystemen ist das Zusammensetzen von kurzen Texten und Meßwerten für die Anzeige auf einem Display oder die Ausgabe auf einem Drucker. Grundlage für einige einfache Stringverkettungen und -funktionen ist der Typ **`string`**. Eine Stringvariable bietet Platz für maximal 30 Zeichen und belegt stets 32 Bytes im Speicher, auch wenn die tatsächlich gespeicherte Zeichenkette kürzer als 30 Zeichen ist.

Ausgaben, die länger als 30 Zeichen sein sollen, können in Bytearray-Variablen aus einzelnen Substrings zusammengesetzt werden (siehe Bibliotheksmodul mem.c2).

5.4.3 Zusammengesetzte Datentypen

Zur Kapselung komplexer Datenstrukturen in einem Typ können aus Standardtypen (`byte` … `string`) und anderen zuvor definierten Typen zusammengesetzte Datentypen gebildet werden. Dazu steht nach dem Schlüsselwort **`type`** der Bezeichner des neuen Datentyps. In geschweiften Klammern folgen die Definitionen der einzelnen Felder des Typs. Die Definition eines Feldes besteht aus dem Schlüsselwort oder Bezeichner eines zuvor bekannten Typs sowie dem Bezeichner des Feldes. Mehrere Felder sind jeweils durch ein Semikolon voneinander getrennt.

Beispiele:

```
type Position
{
  int x;
  int y;
}

type MyType
{
  Position pos;
  float    value;
  string   text;
}
```

Vorteile eigener Typen sind z.B die bessere Lesbarkeit eines Programmes und die einfachere Übergabe zusammengehöriger Daten an Funktionen, also z.B.

```
function fx ( MyType t ) …
```

statt

```
function fx ( int xpos, int ypos,
              float value, string text ) …
```

5.5 Variablen

5.5.1 Definition von Variablen

Variablen dienen zur Zwischenspeicherung von Daten während des Programmablaufes. Vor der ersten Verwendung im Quelltext muß eine Variable durch Angabe des Datentyps und des Bezeichners definiert werden.

```
Typ Name;
```

z.B.

```
int i;
string s;
```

Mehrere Variablen gleichen Typs können in einer gemeinsamen Anweisung definiert werden. Dabei sind mehrere Bezeichner jeweils durch ein Komma voneinander getrennt.

```
Typ Name1, Name2, …;
```

z.B.

```
long x,y,z;
```

Definierte Variablen können nachfolgend in Ausdrücken und Zuweisungen verwendet werden.

z.B.

```
int x, y;
x = 18;
y = 8 * x;
```

5.5.2 Definition und Anwendung von Variablen zusammengesetzter Datentypen

Die Definitionsyntax entspricht der bereits bekannten Syntax für die Definition von Variablen mit Standardtypen. Mit dem Beispiel aus 5.4.3 läßt sich eine Variable vom zusammengesetzten Typ `MyType` wie folgt definieren:

```
MyType t;
```

Danach kann auf die einzelnen Felder der Variablen t durch Anhängen eines Punktes und des jeweiligen Feldbezeichners zugegriffen werden.

```
t.value = 82.5;
```

So sind auch die Felder verschachtelter Datentypen zu erreichen, z.B.:

```
t.pos.x = 31;
```

5.5.3 Definition und Indizierung von variablen Arrays

C2 unterstützt die Definition von variablen eindimensionalen Arrays. Bei der Definition folgt dann nach dem Variablenbezeichner in eckigen Klammern **[ ]** ein konstanter Ausdruck. Der Ergebniswert des Ausdrucks legt die Anzahl der Arrayelemente fest.

```
Typ ArrayName[konstanter Wert];
```

z.B.

```
float coeff[10];
```

Der benötigte Speicherplatz errechnet sich aus der Größe eines einzelnen Elements, multipliziert mit der Anzahl der Elemente. Also werden für das float-Array im obigen Beispiel 80 Bytes belegt (10*8 Bytes).
Der Zugriff auf einzelne Arrayelemente in Ausdrücken und Zuweisungsanweisungen und erfolgt über einen Indexterm in eckigen Klammern. Der Indexterm kann ein beliebiger, auch nichtkonstanter, numerischer Ausdruck sein. Sein Wert wird zur Programmlaufzeit berechnet.
z.B.

```
y[i] = coeff[i] * x[i] + coeff[i-1] * x[i-1];
```

☞ Der Index ist nullbasiert!

D.h. der Indexwert 0 bezieht sich auf das erste Element, der Wert 1 auf das zweite usw.

☞ Während des Programmlaufes erfolgt keine Überprüfung des Index!

Ein häufiger Fehler in Anwenderprogrammen ist die Verletzung des zulässigen Indexbereiches. Das kann vom einfachen lokalen Fehlverhalten des Programm bis zum vollständigen Systemabsturz der C-Control II Unit führen.
Mehrdimensionale Arrays werden in C2 nicht unterstützt. Eine vergleichbare Funktionalität kann über die Verwendung eindimensionaler Arrays von zusammengesetzten Datentypen hergestellt werden.
z.B.

```
Type Line
{
  int row[10];
};
```

```
Type Matrix
{
  Line line[10];
};

Matrix m;
int x;
int i,k;
...
x = m.line[i].row[k];
```

Diese Syntax ist zwar etwas schreibaufwendiger als ein vergleichbares `m[i][k]` in anderen Programmiersprachen, dafür ist die Lesbarkeit von C2-Programmen an dieser Stelle wesentlich besser.

5.5.4 Initialisierung

Der Wert einer Variablen nach der Definition ist zunächst unbestimmt. Bevor eine Variable zur Berechnung eines Ausdrucks herangezogen wird, sollte sie initialisiert werden. Anderenfalls ist auch das Ergebnis des Ausdrucks unbestimmt (abgesehen von trivialen Ausdrücken, wie 0*x). Die Initialisierung erfolgt durch Wertzuweisung,

z.B.

```
int i;
i = 0;
```

Bei Arrayvariablen muß jedes Element einzeln initialisiert werden, z.B. in einer Schleife, die den Index von 0 bis zur Anzahl der Elemente - 1 laufen läßt.

```
int i;
long table[13];

for i=0 ... <13
  table[i] = 0;
```

Variablen zusammengesetzter Datentypen sind erst dann vollständig initialisiert, wenn alle einzelnen Felder, auch die Felder verschachtelter Typen initialisiert sind.

```
MyType t;

t.pos.x = 0;
t.pos.y = 0;
t.value = 0;
t.text = "";
```

5.5.5 Globale und lokale Variablen

C2 und die virtuelle Maschine der C-Control II Unit unterscheiden zwischen globalen und lokalen Variablen.
Globale Variablen werden auf Modulebene neben Threads, Funktionen, benannten Konstanten und zusammengesetzten Datentypen definiert.

z.B.

```
int i;

function fx ()
{
  //...
}
```

Globale Variablen existieren während des gesamten Programmablaufes statisch an einer ganz bestimmten, vom Compiler berechneten Speicherstelle. Über den Modulnamen sind sie im gesamten Quelltext nach der Definition sichtbar und zugreifbar. Globale Variablen sollten sehr sparsam und gut überlegt verwendet werden! Nach Möglichkeit sollten sämtliche Manipulationen einer Variablen innerhalb desselben Moduls vorgenommen werden, in dem sie definiert wurde. Anderenfalls wird ein Programm schnell unübersichtlich, wenn nicht mehr klar erkennbar ist, an welcher Stelle sich der Wert einer globalen Variablen ändern kann. Typische Anwendung für globale Variablen sind Variablen zur Speicherung von Programmzuständen und Benutzereinstellungen, Variablen für den Datenaustausch zwischen Threads oder Bytearray-Variablen als Pufferspeicher bei einer Datenübertragung. Lokale Variablen werden im Anweisungsblock eines Threads oder einer Funktion definiert.

z.B.

```
function fx ()
{
  int i;

  //...
}
```

Lokale Variablen einer Funktion werden zur Programmlaufzeit auf dem Stack des aktuellen Threads angelegt und existieren nur innerhalb eines Speicherkontextes, d.h. während der Abarbeitung einer Funktion. Sie sind nur innerhalb dieses Kontextes sichtbar und zugreifbar. Beim Verlassen einer Funktion endet der Lebenszyklus einer lokalen Variable. Beim Wiedereintritt in diese Funktion oder dem parallelen Aufrufen der Funktion durch einen anderen Thread ist der Wert einer lokalen Variable stets unbestimmt.
Eine Sonderform stellen lokale Variablen von Threads dar. Da der Speicherkontext von Threads während des gesamten Programmlaufes bestehen bleibt, auch für angehaltene Threads, existieren die lokalen Variablen eines Threads quasi-statisch. Ein Bytearray könnte somit auch als Pufferspeicher für Datenübertragungen verwendet werden (siehe Bibliotheksmodul hwcom.c2).

z.B.

```
thread tx
{
  byte buf [48];

  //...
}
```

Bei der Vergabe von Namen für lokale Variablen ist zu beachten, daß sie eventuell globale Bezeichner desselben Moduls verdecken. Will man dann auf gleichnamige globale Bezeichner zugreifen, muß zusätzlich der Modulname spezifiziert werden, als würde sich der globale Bezeichner in einem anderen Modul befinden.

z.B. in Modul a.c2

```
int i;

function fx ()
{
  int i;

  i = 0;    // <- lokales i
  a.i = 0;  // <- globales i
}
```

5.6 Konstanten

5.6.1 Benannte und unbenannte Konstanten

Unbenannte Konstanten werden sehr häufig verwendet. In der Anweisung

```
a = 1;
```

ist "1" eine unbenannte Zahlenkonstante.

Benannte Konstanten repräsentieren einen Wert, der ihnen zuvor in der Konstantendefinition (siehe weiter unten) zugewiesen wurde. Die Definition und Verwendung von benannten Konstanten hat folgende Vorteile:

- Reduzierung des Aufwandes bei eventuellen Änderungen im Programm - Konstanten müssen nur an der Stelle ihrer Definition modifiziert werden, nicht an den vielen Stellen ihrer Verwendung im Programm.
- Erhöhung der Lesbarkeit eines Programmes, wenn Konstanten mit vollständig selbstbeschreibenden Bezeichnern definiert werden (also z.B. ERDUMFANG statt EUMF)
- für wiederholte Verwendung derselben `long`-, `float`- und `string`-Konstanten wird weniger Speicherplatz im Konstantenspeicher benötigt. Unbenannte Konstanten mit gleichem Wert oder Textinhalt würden nämlich bei mehrfacher Verwendung im Programmquelltext mehrfach im Konstantenspeicher angelegt werden.

Konstanten mit zusammengesetztem Datentyp werden in C2 nicht unterstützt.

5.6.2 Unbenannte Zahlenkonstanten

Dezimalzahlen bestehen aus einer Folge der Ziffern 0 … 9 ohne Zwischenräume. Optional kann ein Minus als negatives Vorzeichen vorangestellt werden.
Bei dezimalen **Fließkommazahlen** folgen ohne Zwischenraum ein Dezimalpunkt (kein Komma!) und die Nachkommastellen. Das Exponentialformat wird nicht unterstützt.
Hexadezimalzahlen sind Folgen der Hexadezimalziffern 0 … 9, A … F bzw. a … f mit dem Präfix "0x" oder "0X" (vgl. C/C++).
Binärzahlen sind Folgen der Binärziffern 0 und 1 mit dem Präfix "0b" oder "0B".
Oktalzahlen werden nicht unterstützt.

Beispiele:

Dezimalzahlen	0	17	-12345
Fließkommazahlen	0.0	1.5	-123.456
Hexadezimalzahlen	0x0	0xFF	0XABCD
Binärzahlen	0b0	0b01	0B11101

5.6.3 Unbenannte Zeichenkonstanten

Zeichenkonstanten stehen für deren ASCII-Codes (Wertebereich 0 … 255) und können wie ganze Zahlen in numerischen Ausdrücken verwendet werden. Unbenannte Zeichenkonstanten sind von zwei Hochkommata eingeschlossen und bestehen selbst aus einem einzelnen Zeichen oder einem Sondercode.
Sondercodes ermöglichen die Darstellung von Zeichen, die im Quelltext nicht sichtbar wären (z.B. Steuerzeichen, Zwischenraumzeichen). Auch das Hochkomma selbst muß als Sondercode geschrieben werden. Sondercodes beginnen mit einem Backslash ****. Anschließend folgt ohne Zwischenraum ein Codezeichen (nicht case-sensitiv) oder die Angabe eines ASCII-Codes als Dezimal- oder Hexadezimalzahl, der Hexadezimalpräfix ist hier nur ein x ohne 0.

Sondercodes mit Codezeichen:

Codezeichen	Bedeutung	Vollständige Zeichenkonstante
a	Klingelton (bell)	'\a'
b	Backspace (ein Zeichen zurück)	'\b'
f	Steuerzeichen "form feed", Seitenvorschub auf einem Ausgabegerät	'\f'
n	Steuerzeichen "new line", Zeilenvorschub auf einem Ausgabegerät	'\n'
r	Steuerzeichen "carriage return", Wagenrücklauf auf einem Ausgabegerät	'\r'
t	Tabulator (Zwischenraumzeichen)	'\t'
\	Backslash	'\\'
'	Hochkomma	'\''
"	Anführungszeichen	'\"'

Sondercodes mit ASCII-Code (Beispiele):

Zeichen	Konstante in Dezimalform	Konstante in Hexadezimalform
A	'\65'	'\x41'
a	'\97'	'\x61'
0	'\48'	'\x30'
$	'\36'	'\x24'
Tabulator	'\9'	'\x9'

5.6.4 Unbenannte Stringkonstanten

Konstante Strings (oder "Zeichenketten") sind konstante Texte in zwei Anführungszeichen "". Zwischen den Anführungszeichen kann jedes darstellbare Zeichen stehen. Soll der String selbst ein Anführungszeichen enthalten, so muß dieses per Sondercode (siehe oben) eingebettet werden. Das gilt auch für nicht darstellbare Steuerzeichen. Der konstante String

```
"\"abc\txyz\""
```

enthält also 9 Zeichen: ein Anführungszeichen, die Buchstabenfolge "abc", einen Tabulator, die Buchstabenfolge "xyz" und noch ein Anführungszeichen.

☞ Das abschließende Anführungszeichen einer Stringkonstanten muß vor dem Zeilenende stehen.

Längere Strings können gebildet werden, in dem zwei Stringkonstanten, jeweils in Anführungszeichen, hintereinander im Quelltext stehen. Zwischen den Teilstrings dürfen beliebige Zwischenraumzeichen stehen, auch Zeilenvorschübe.

```
"abc"
  "xyz"
```

wird vom C2-Compiler verkettet zu

```
"abcxyz"
```

5.6.5 Definition von benannten Konstanten

Benannte Konstanten werden stets global auf Modulebene definiert. Sie sind nach der Definition über die Angabe des Modulbezeichners im gesamten Programm verfügbar (vgl. globale Variablen). Lokale Konstanten von Threads und Funktionen gibt es nicht.
Die Definition einer benannten Konstante beginnt stets mit dem Schlüsselwort **`const`**. Anschließend folgen der Bezeichner, ein Zuweisungsoperator sowie ein konstanter Ausdruck und abschließend ein Semikolon.

```
const Name = konstanter Ausdruck;
```

z.B.:

```
const A = 1000;
const B = A + 100000;
const C = 17.4;
```

Der Datentyp einer Konstanten wird vom Compiler automatisch bestimmt. Es wird der maximal notwendige Typ verwendet. Für `A` im obigen Beispiel ist maximal ein `int`-Datentyp notwendig, für `B` jedoch ein `long`, und `C` kann nur durch einen `float`-Typ dargestellt werden. `B` und `C` werden vom Compiler im Konstantenspeicher der C-Control II Unit angelegt, wo sie eine feste Adresse haben. Spezielle Operationscodes der virtuellen Maschine der C-Control II Unit laden benannte und unbenannte Byte- und Integerkonstanten, wie oben `A`, immer immediat, d.h. eingebettet in den Operationscode. Für `A` wird daher kein Platz im Konstantenspeicher belegt.

Neben numerischen Konstanten können auch Stringkonstanten benannt werden. Statt des konstanten numerischen Ausdrucks muß dann eine Zeichenkette in Anführungszeichen nach dem Zuweisungsoperator stehen.

```
const Name = "Text";
```

z.B.:

```
const GREETINGS = "Hallo C2";
const TABLEHEAD = "Nummer\tZeit\tWert";
```

Stringkonstanten belegen im Konstantenspeicher der C-Control II Unit nur so viele Bytes, wie sie Zeichen enthalten, zuzüglich eines Bytes zur Speicherung der Stringlänge. Im Gegensatz zu `string`-Variablen können Stringkonstanten auch mehr als 30 Zeichen enthalten. Sie werden jedoch bei Stringoperationen auf maximal 30 Zeichen reduziert.

5.6.6 Benannte konstante Arrays

Sowohl von numerischen Konstanten als auch von Stringkonstanten lassen sich benannte eindimensionale Arrays anlegen. In beiden Fällen steht nach dem Bezeichner ein Paar eckiger Klammern [ ]. Nach dem Zuweisungsoperator werden, jeweils durch ein Komma getrennt, die einzelnen Elemente aufgelistet. Die Größe des Arrays ergibt sich automatisch aus der Zählung der aufgelisteten Elemente. Ein Array kann auch aus nur einem Element bestehen.

```
const Name[] = element1, element2 ...;
```

z.B.:

```
const CHARACTERS[] = 'A', 'B', 'C';
const TABLE1[] = 0, 100, 10000, 1000000;
const TABLE2[] = 0, 1.5, 3, 17;
const ONE[] = 1;
```

Arrays von numerischen Werten werden automatisch in dem maximal notwendigen Zahlenformat abgelegt. Dieses ergibt sich aus dem Element mit dem anspruchsvollsten Datentyp. So wird `TABLE1` im obigen Beispiel ein Array von `long`-Werten (wegen des Elementes 1000000), `TABLE2` wird ein `float`-Array (wegen des Elementes 1.5).
`CHARACTERS` und `ONE` sind Integer-Arrays.
Wie Arrays von numerischen Konstanten können Arrays von Stringkonstanten definiert werden:

z.B.:

```
const menu[] = ”rice”, ”couscous”, ”potatos”;
```

5.7 Operatoren

5.7.1 Rangfolge

Operatoren teilen numerische Ausdrücke in Teilausdrücke. Dabei werden die Operatoren in einer von ihrem Rang abhängigen Reihenfolge ausgewertet und die Teilausdrücke zur Programmlaufzeit nacheinander berechnet (vgl. Vereinbarung in der Mathematik ”Punktrechnung vor Strichrechnung”).

z.B.:

```
a = 10 + 4 * 2;     // a wird 18
```

Ausdrücke mit Operatoren gleichen Ranges werden von links nach rechts berechnet.

z.B.:

```
a = 10 / 4 / 2;     // a wird 1,25
```

Wie aus der Mathematik bekannt ist, kann die Rechenreihenfolge durch Klammersetzung beeinflußt werden

z.B.:

```
a = 10 / (4 / 2);  // a wird 5
```

Klammerebenen können theoretisch beliebig tief ineinander verschachtelt werden. Allerdings geht in der Regel bereits ab der dritten oder vierten Verschachtelung jeglicher Überblick über den dargestellten Ausdruck verloren. Außerdem können extrem tiefe Verschachtelungen zu Stackproblemen führen (siehe 4.4.6). Das Programm arbeitet dann nicht korrekt. Teilen Sie daher die Berechnung komplexer Ausdrücke nach Möglichkeit in mehrere Anweisungen, und speichern Sie Zwischenergebnisse in lokalen Variablen. Fügen Sie die Zwischenergebnisse nacheinander zum Endergebnis zusammen.

Auch wenn die Rangfolge es nicht erfordern würde, kann eine zusätzliche Klammersetzung um Teilausdrücke die Lesbarkeit des Quelltextes erhöhen,

z.B.:

```
(x > 10) & (x < 20)
```

statt

```
x > 10 & x < 20
```

Rangfolge der Operatoren in C2:

Rang	Operator
8	( )
7	- (negatives Vorzeichen) ! not
6	* / % mod
5	+ -
4	<< shl >> shr
3	== != > < >= <=
2	& and !& nand
1	| or !| nor ^ xor

Zu einigen Operatoren existiert neben einem Symbol eine alternative Schlüsselwortform, z.B. stehen % und mod für die Modulodivision. Wählen Sie selbst, welche Form Sie bevorzugen, das Ergebnis bleibt gleich.

5.7.2 Arithmetische Operatoren

Operator	Bedeutung	Beispielausdruck	Ergebnis
+	Addition	1 + 1	2
-	Subtraktion	2 - 1	1
*	Multiplikation	2 * 3	6
/	Division	6 / 3 18 / 4 18.0 / 4	2 4 4,5
% mod	Modulodivision (Divisionsrest)	18 mod 4 18 % 4 1.8 % 0.4	2 2 0,2
-	negatives Vorzeichen	-(1+1)	-2

5.7.3 Bitschiebeoperatoren

Operator	Bedeutung	Beispielausdruck	Ergebnis
<< shl	links schieben	1 << 1 3 shl 2	2 12
>> shr	logisch rechts schieben	1 >> 1 5 shr 2 -1 shr 1 (long) -1 shr 1	0 1 32767 2147483647

5.7.4 Vergleichsoperatoren

Vergleichsoperatoren liefern den Wert -1, (minus 1, nicht 1!), falls der Ausdruck wahr ist. Ist der Ausdruck falsch, wird das Vergleichsergebnis 0. Der Wert -1 entspricht hexadezimal dem Integerwert 0xFFFF bzw. dem Longinteger 0xFFFFFFFF.

Operator	Bedeutung	Beispielausdruck	Ergebnis
==	ist gleich?	1 == 1	-1
		1 == 2	0
!=	ist ungleich?	1 != 1	0
		1 != 2	-1
>	ist gößer?	2 > 1	-1
		1 > 2	0
<	ist kleiner?	2 < 1	0
		1 < 2	-1
>=	ist größer oder gleich?	2 >= 1	-1
		1 >= 1	-1
		1 >= 2	0
<=	ist kleiner oder gleich?	2 <= 1	0
		1 <= 1	-1
		1 <= 2	-1

5.7.5 Logische Operatoren und Bitmanipulationen

In C2 sind logische Verknüpfungen immer Bitoperationen. Es wird nicht wie beispielsweise in C/C++ in Bit-AND und logisches AND unterschieden.

Operator	Bedeutung	Beispielausdruck	Ergebnis
! not	nicht (Bitinvertierung)	!1	-2
		not 0	-1
		not 2.5	-0
		not 2.0	-3
		!(2 < 1)	-1
		!(1 < 2)	0

Operator	Bedeutung	Beispielausdruck	Ergebnis
& and	und	1 & 1	1
		1 and 0	0
		14 & 3	2
		(1<2)&(2<3)	-1
		(1<2)&(3<2)	0
!& nand	und mit anschließender Bitinvertierung	1 !& 1	-2
		1 nand 0	-1
		14 !& 3	-3
		(1<2)!&(2<3)	0
		(1<2)!&(3<2)	-1
| or	oder	1 | 1	1
		1 or 0	1
		0 or 0	0
		14 or 1	15
		(1<2)|(2<3)	-1
		(1<2)|(3<2)	-1
		(2<1)|(3<2)	0
!| nor	oder mit anschließender Bitinvertierung	1 !| 1	-2
		1 nor 0	-2
		0 nor 0	-1
		14 nor 1	-16
		(1<2)|(2<3)	0
		(1<2)|(3<2)	0
		(2<1)|(3<2)	-1
^ xor	exklusiv-oder	1 ^ 1	0
		1 xor 0	1
		0 xor 0	0
		14 ^ 3	3
		(1<2)^(2<3)	0
		(1<2)|(3<2)	-1
		(2<1)|(3<2)	-1

Eine Besonderheit bilden logische Operationen mit `float`-Operanden. Hier findet vor der logischen Verknüpfung eine automatische Konvertierung in einen Integerwert 0 oder -1 statt: der float-Wert 0.0 wird zum Integer 0, alle Werte ungleich 0.0 werden zu -1.

Das gilt jedoch nicht für konstante `float`-Ausdrücke, die keinen "echten" Fließkommawert haben, z.B. 2.0, da diese vom Compiler als `int`- oder `long`-Konstanten betrachtet werden.

z.B.:

```
float x;
int result;

x = 2;
result = not x;    // result wird 0
result = not 2.0;  // result wird -3
```

5.7.6 Stringverkettung mit dem Operator +

Bei Zuweisungen an `string`-Variablen kann auf der rechten Seite des Zuweisungsoperators ein verketteter Stringausdruck stehen. In der Verkettung werden Teilstrings zu einem Ergebnis zusammengefügt. Die Teilstrings sind jeweils durch einen +-Operator voneinander getrennt. Als ein einzelner Teilstring kann

- ein Bezeichner einer `string`-Variable
- ein indizierter Bezeichner eines variablen Stringarrays
- ein Bezeichner einer `string`-Konstante
- ein indizierter Bezeichner einer konstanten Stringarrays
- eine unbenannte Stringkonstante
- ein numerischer Ausdruck

stehen. Numerische Ausdrücke in der Stringverkettung müssen in Klammern stehen, wenn sie selbst Operatoren enthalten. Ein numerischer Ausdruck wird als ASCII-Code eines Zeichens interpretiert und als solches im Ergebnisstring eingebunden. Gegebenenfalls erfolgt eine Reduzierung des Wertes auf den Bereich von 0 … 255.
Verkettungen werden automatisch auf maximal 30 Zeichen begrenzt.
Beispiel für eine Stringverkettung mit +:

```
const S = "AAA";
const SA[] = "XXXX", "YYYY", "ZZZZ";

string s1;
string sa[3];
string s;
```

```
s1    = ”bbb”;
sa[0] = ”uuu”;
sa[1] = ”vvv”;
sa[2] = ”www”;

// Stringzuweisung mit Verkettung:
s = s1 + sa[2] + S + SA[0] + ”ccc” + (’A’+3);
```

Nach dieser Anweisung enthält s den Text ”bbbwwwAAAXXXXcccD”.

5.8 Funktionen

Die virtuelle Maschine der C-Control II Unit unterstützt die Programmierung mit Unterfunktionen. Blöcke von Anweisungen, die im Programm mehrfach benutzt werden, können in Funktionen zusammengefaßt werden. Beim Aufruf einer Funktion können Parameter übergeben werden. Die Funktion selbst kann ein Rechenergebnis zurückgeben. In C2 gibt es keine Trennung zwischen Deklaration und Definition einer Funktion. Ist eine Funktion einmal in einem Modul geschrieben, kann sie weiter unten in diesem Modul und allen nachfolgenden Modulen des Projektes verwendet, also aufgerufen werden. Der Quelltext einer Funktion besteht aus dem Funktionskopf und einem Anweisungsblock.

```
function fx ()
{
  //... Anweisungen
}
```

5.8.1 Funktionskopf

Der Funktionskopf beginnt mit dem Schlüsselwort **`function`**. Anschließend folgen der Funktionsname (Bezeichner) und in runden Klammern die Liste der formalen Parameter. Optional kann dann nach dem Schlüsselwort **`returns`** ein Ergebnistyp spezifiziert werden.

```
function name(type1 name1, ...) returns type
```

Funktionen in C2 können nur numerische Ergebnisse zurückgeben, also die Datentypen `byte, int, long` und `float`. Die Rückgabe von Strings, Arrays oder zusammengesetzten Typen ist nicht möglich.

```
function fx() returns byte     // OK

function fx() returns int      // OK

function fx() returns long     // OK

function fx() returns float    // OK

function fx() returns string   // Fehler!

function fx() returns MyType   // Fehler!
```

In der Liste der formalen Parameter werden Typen und Namen der beim Aufruf zu übergebenden Daten spezifiziert. Mehrere Parameter sind jeweils durch ein Komma voneinander getrennt. Hat eine Funktion keine Parameter, stehen nach dem Funktionsnamen nur die öffnende und schließende runde Klammer. Als Parameter können numerische Daten, Strings oder Daten mit zusammengesetztem Typ übergeben werden. Auch Arrays sind möglich. Bei der Übergabe von Arrays erfolgt keine Größenangabe in den eckigen Klammern nach dem Parameternamen.

Beispiele:

· Funktion mit einem Integerparameter und einem Integerergebnis

```
function fx( int x ) returns int
```

· Funktion mit einem Stringparameter

```
function fx( string s )
```

· Funktion mit einem long-Array und einem Integerparameter

```
function fx( long a[], int i )
```

· Funktion mit einem benutzerdefinierten MyType-Parameter

```
function fx( MyType t )
```

5.8.2 Parameter und lokale Variablen

Im Anweisungsblock einer Funktion können lokale Variablen definiert werden (siehe 5.5.5) Die im Funktionskopf definierten Parameter können ebenso wie Variablen verwendet werden. Numerische Parameter (byte … float) sind echte lokale Variablen der Funktion. Sie werden beim Aufruf der Funktion auf dem Stack des aktuellen Threads gespeichert und mit dem übergebenen Wert initialisiert.

Variable Strings, Arrays und Parameter mit anwenderdefiniertem Datentyp hingegen werden automatisch als Referenz übergeben. Manipulationen an Referenzparametern wirken sich auf das referenzierte Datenobjekt aus.

z.B.

```
function fx ( string s )
{
  s = ”abc”;
}

thread main
{
  string local_s;

  local_s = ”123”;
  fx(local_s);     // local_s wird ”abc”
}
```

im Gegensatz zu numerischen Parametern:

```
function fx ( int i )
{
  i = 0;
}

thread main
{
  int local_i;

  local_i = 1;
  fx(local_i);     // local_i bleibt 1
}
```

5.8.3 Ende einer Funktion und Ergebnisrückgabe

Eine Funktion endet automatisch, wenn die Programmausführung zur schließenden geschweiften Klammer des Anweisungsblocks gelangt. Eine Funktion mit Rückgabewert liefert dann das Ergebnis 0.

z.B.:

```
function fx () returns int
{
}

thread main
{
  int i;
  i = 1;
  i = fx();  // i wird 0
}
```

Mit der **return-**Anweisung kann eine Funktion vorzeitig beendet werden und den Wert eines numerischen Ausdrucks als Ergebnis zurückgeben,

```
return;
return numerischer Ausdruck;
```

Die erste Form darf nur für Funktionen ohne definierten Rückgabetyp verwendet werden. Die zweite Form mit numerischem Ausdruck ist für Funktionen mit definiertem Rückgabetyp reserviert.

z.B.:

```
function fx ( int param ) returns int
{
  return param * param + 100;
}
```

5.8.4 Aufruf

Der Aufruf einer Funktion erfolgt durch Angabe ihres Bezeichners, gefolgt von einer öffnenden und einer schließenden runden Klammer. Wenn im Kopf der Funktion formale Parameter definiert wurden, so müssen beim Funktionsaufruf innerhalb der runden Klammern genau so viele aktuelle Parameter aufgelistet werden, jeweils durch ein Komma getrennt.

z.B.:

```
function fx ( int a, int b )   // zwei Parameter
{
  //...
}

fx();                          // Fehler!
fx(17);                        // Fehler!
fx(17,4);                      // OK
```

Aufrufe von Funktionen, die einen Rückgabewert liefern, können in numerischen Ausdrücken verwendet werden, aber auch als einzelne Anweisung. Aufrufe von Funktionen ohne Rückgabewert dürfen ausschließlich als einzelne Anweisung stehen.

z.B.:

```
function get_something () returns int
{
  //...
}

function do_something ()
{
  //...
}

get_something();       // OK, Rückgabewert ignoriert
do_something();        // OK

int a;
a = get_something();   // OK
a = do_something();    // Fehler!
```

5.8.5 Typenprüfung

Der C2-Compiler führt zu jedem Aufruf einer Funktion eine Prüfung durch, ob neben der Anzahl der Parameter auch deren jeweiliger Typ der Funktionsdefinition entspricht. Ein Bezeichner einer `string`-Variablen kann z.B. nicht übergeben werden, wenn laut Definition an dieser Stelle ein numerischer Ausdruck erwartet wird.

Es gelten folgende Typkompatibilitätsregeln:

Typ des formalen Parameters im Funktionskopf	zulässige aktuelle Parameter beim Funktionsaufruf
`byte, int, long, float`	beliebiger numerischer Ausdruck
`byte[]`	Bezeichner einer `byte`-Arrayvariable*), Bezeichner einer `string`-Variable, indizierter Bezeichner einer `string`-Arrayvariable
`int[]`	Bezeichner einer `int`-Arrayvariable*)
`long[]`	Bezeichner einer `long`-Arrayvariable*)
`float[]`	Bezeichner einer `float`-Arrayvariable*)
`string`	unbenannte `string`-Konstante, Bezeichner einer `string`-Variable, indizierter Bezeichner einer `string`-Arrayvariable, Bezeichner einer `string`-Konstante, indizierter Bezeichner einer `string`-Arraykonstante
`string[]`	Bezeichner einer `string`-Arrayvariable*)
`zusammengesetzter Typ, z.B. MyType`	Bezeichner einer `MyType`-Variable, indizierter Bezeichner einer `MyType` -Arrayvariable
`MyType[]`	Bezeichner einer `MyType`-Arrayvariable *)

*) Referenzen auf konstante Arrays dürfen nicht an Funktionen übergeben werden.

5.8.6 Rekursion

Eine Funktion kann sich theoretisch auch selbst aufrufen. Das wird als Rekursion bezeichnet. Einige mathematische Näherungen beruhen auf rekursiven Algorithmen mit Abbruchschranken.

☞ C2 verbietet rekursive Funktionsaufrufe nicht ausdrücklich, sie sollten jedoch vermieden werden.

Rekursionen führen ab einer gewissen Tiefe immer zur Überschreitung des Stackbereiches, der für einen Thread reserviert ist. Dann werden Daten anderer Threads ungewollt überschrieben. Das kann von lokalem Fehlverhalten bis zum Absturz des gesamten Systems der C-Control II Unit führen!

5.8.7 Inline-Funktionen und -Anweisungen

Die Bibliotheksmodule zum Zugriff auf die Ressourcen der C-Control II Unit (z.B. hwcom.c2) benutzten **`inline`**-Funktionen und -Anweisungen zum direkten Einfügen von virtuellen Maschinencodes in den Programmquelltext. Es gilt, **`inline`**-Funktionen dürfen nur **`inline`**-Anweisungen enthalten. In einer **`inline`**-Anweisung muß nach dem Schlüsselwort ein konstanter Ausdruck stehen, der einen Operationscode darstellt (siehe 4.4.2).

☞ inline-Funktionen und -Anweisungen werden nur von C2-Systementwicklern benötigt. Eine ausführliche Dokumentation der einzelnen Operationscodes der virtuellen Maschine ist nicht Bestandteil des Lieferumfanges der C-Control II Unit.

5.9 Threads

5.9.1 Definition

Threads werden auf Modulebene definiert und sind nach der Definition global sichtbar. Es gibt keine verschachtelten Threads innerhalb von Threads und keine lokalen Threads innerhalb von Unterfunktionen. Die Definition eines Threads beginnt mit dem Schlüsselwort **`thread`**. Dann folgen der Bezeichner und ein Anweisungsblock in geschweiften Klammern.

```
thread Name
{
  // Anweisungen
}
```

Innerhalb des Anweisungsblocks können lokale Variablen definiert werden, die quasistatisch sind (siehe 5.5.5). Der gesamte Code des Anweisungsblock wird automatisch in einer Endlosschleife ausgeführt.

Beispiel:

```
thread blink2
{
  ports.set(2,-1);
  sleep 200;
  ports.set(2,0);
  sleep 800;
}
```

5.9.2 main-Threads

Ein Thread, dessen Bezeichner nicht "main" ist, hat zum Programmstart die Priorität 0, d.h. er befindet sich im Stillstand, seine Anweisungen werden nicht ausgeführt.
Jedes Modul kann einen main-Thread enthalten, also einen Thread mit dem Bezeichner "main". Dieser hat bei Programmstart die Standardpriorität 32. Seine Anweisungen werden von Beginn an ausgeführt. Die Aufgabe der main-Threads ist es, Initialisierungen vorzunehmen und bei Bedarf andere Threads zu starten.
Ein Programm sollte mindestens ein Modul mit einem main-Thread haben. Anderenfalls steht das gesamte Programm still und wartet endlos auf den run-Befehl (siehe 5.9.3).

5.9.3 Prioritätssteuerung

Die virtuelle Maschine der C-Control II Unit stellt jedem Thread soviel Rechenkapazität zur Verfügung, wie es seinem Prioritätswert entspricht. Ein Thread mit Priorität 32 kann genau 32 virtuelle Maschinenoperationen hintereinander ausführen, bevor ihn das System unterbricht und der nächste Thread an der Reihe ist. Zur Orientierung: die Anweisung

```
a = b + c;
```

wird in vier virtuellen Maschinenoperationen ausgeführt, wenn a,b und c vom gleichen numerischen Datentyp sind:

```
1. b auf den Stack laden
2. c auf den Stack laden
3. Addition
4. Ergebnis in a speichern
```

Komplexere Anweisungen sind entsprechend umfangreicher. Das System kann einen Thread durchaus auch innerhalb einer Anweisung unterbrechen, z.B. vor der Addition im obigen Beispiel. Da jeder Thread mit seinem eigenen Stack arbeitet, gibt es dabei keine Probleme.
Zur Änderung der Priorität eines Threads gibt es in C2 verschiedene Schlüsselworte.

- **run**

Das Schlüsselwort **run** wird in zwei Formen verwendet. Form 1 setzt die Priorität des angegebenen Threads auf den Standardwert 32. Diese Form dient in der Regel dazu, um von main-Threads aus andere Threads zu starten. Form 2 setzt die Priorität des aktuell ausgeführten Threads auf das Ergebnis des angegebenen numerischen Ausdrucks.

Form 1:

```
run ThreadName;
```

z.B.:

```
run blink2;
```

Form 2:

```
run numerischer Ausdruck;
```

z.B.:

```
run 100;
```

Beachten Sie, daß ein stillstehender Thread (Priorität 0) sich niemals mit `run` selbst starten kann!

- **halt**

Die **halt**-Anweisung setzt die Priorität eines Threads auf 0. Es gibt zwei Formen. Mit Form 1 kann ein beliebiger laufender Thread einen anderen Thread oder auch sich selbst anhalten. Form 2 bezieht sich immer auf den aktuell ausgeführten Thread.

Form 1:

```
halt ThreadName;
```

z.B.:

```
halt blink2;
```

Form 2:

```
halt;
```

Angehaltene Threads können nur durch andere Threads wieder gestartet werden.

- **resume**

Die **resume**-Anweisung setzt die Priorität eines Threads auf den Wert vor der letzten `run-` oder `halt-`Anweisung in Bezug auf diesen Thread. Es gibt ebenfalls zwei Formen. Form 1 bezieht sich auf den angegebenen Thread, Form 2 auf den aktuell laufenden.

Form 1:

```
resume ThreadName;
```

z.B.:

```
halt blink2;    // blink2 steht
resume blink2;  // blink2 läuft wie vor dem halt
```

Form 2:

```
resume;
```

z.B.:

```
run 100;   // aktueller Thread läuft mit Prio. 100
resume;    // aktueller Thread läuft wie vor run 100
```

- **yield**

Mit Ausführung der **yield**-Anweisung gibt der aktuelle Thread die Programmausführung, unabhängig von seiner Priorität, sofort an den nächsten Thread ab.

```
yield;
```

Der **yield** -Befehl wird relativ selten benötigt.

5.9.4 Warten auf Ereignisse

In bestimmten Situationen soll ein Thread auf das Eintreten eines Ereignisses warten und in der Wartephase möglichst wenig Rechenkapazität belegen. Dafür dient in C2 die **wait**-Anweisung.

Die **wait**-Anweisung prüft den Wert eines angegebenen numerischen Ausdrucks. Ist der Wert gleich 0, dann gibt der aktuelle Thread die Programmausführung, unabhängig von seiner Priorität, sofort an den nächsten Thread ab (vgl. yield). Dadurch wird vermieden, daß Threads mit hoher Priorität wartend das System blockieren.

Ein Thread wiederholt die **wait**-Anweisung so lange, bis der numerische Wert ungleich 0 wird.

```
wait numerischer Ausdruck;
```

z.B.:

```
wait ports.get(3); // wartet auf High-Pegel an Port3
```

5.9.5 Pausen

In vielen Anwendungen ist es erforderlich, daß ein Thread seine Ausführung für eine bestimmte Zeit unterbricht und danach automatisch weiterläuft. Dazu gibt es in C2 die **sleep**-Anweisung. Nach dem Schlüsselwort **sleep** folgt ein numerischer Ausdruck.

```
sleep numerischer Ausdruck;
```

z.B.:

```
sleep 1000;
```

Der berechnete Wert des numerischen Ausdrucks bestimmt die Ruhepause des aktuellen Threads in Millisekunden. Der Wert des Ausdrucks bleibt auf den `int`-Bereich beschränkt. Gegebenenfalls nimmt die virtuelle Maschine eine Konvertierung vor. Negative Pausenwerte von -1 bis -32768 werden als Werte von 65535 bis 32768 interpretiert (Zweierkomplement).

Die Prüfung, ob eine Pause beendet ist, erfolgt mit jedem Zyklus, in dem der Thread Rechenzeit erhält. In einer Anwendung mit sehr vielen Threads, die mit hohen Prioritäten lange Umlaufzeiten nach sich ziehen, kann die tatsächliche Pause daher etwas länger als ursprünglich spezifiziert werden.

Ist die Pause noch nicht vorüber, dann gibt der aktuelle Thread die Programmausführung sofort an den nächsten Thread ab (vgl. `yield`).

5.9.6 Synchronisation

In Computersystemen mit parallelen Prozessen kann es zu folgenden problematischen Situationen kommen:

- Aliasing von Speicherzugriffen
- Konkurrenz mehrerer Prozesse um eine Ressource

Konkurrenz um eine Ressource entsteht beispielsweise, wenn zwei Threads gleichzeitig Daten über dieselbe serielle Schnittstelle senden wollen. Die serielle Schnittstelle kann aber nur einen Ausgabepuffer zu einer Zeit bedienen. Folglich kann nur ein Thread gleichzeitig senden. Der zweite Thread muß warten, bis die Ressource frei ist.
Zur Erläuterung des Aliasing-Problems folgendes Beispiel: Ein Thread fragt zyklisch zwei Meßkanäle ab und speichert diese in zwei globalen Variablen. Ein paralleler Thread liest diese globalen Variablen und soll jeweils beide Werte aus einem Meßzyklus einer Prüffunktion zuführen.

```
float a, b;  // Messwerte

thread measure
{
  a = get_channel_a();
  b = get_channel_b();
  //...
}

thread watch
{
  check(a, b);
  // ...
}
```

Wie in 5.9.3 bereits gezeigt, wird eine C2-Anweisung in mehreren Einzeloperationen der virtuellen Maschine ausgeführt. Die Anweisung

```
check(a, b)
```

führt etwa zu folgender Befehlskette:

```
1. a auf den Stack laden
2. b auf den Stack laden
3. Funktion check aufrufen
```

Wie in 5.9.3 ebenfalls erläutert wurde, kann diese Befehlskette jederzeit von einem Threadwechsel unterbrochen werden, also auch zwischen dem 1. und 2. Befehl im obigen Beispiel. Dann kann der `measure-`Thread bereits neue Werte erfaßt haben, bevor der `watch-`Thread wieder an die Reihe kommt. Der `watch-`Thread setzt seine Ausführung mit dem Laden von `b` und dem Aufruf von `check` fort. An die Funktion `check` wird jetzt also ein `a` aus einem alten und `b` aus dem neuen Meßzyklus übergeben. Abhängig davon, was `check` konkret von `a` und `b` erwartet, kann das zu einer schweren Fehlfunktion des Programms führen! Im ungünstigsten Fall läuft ein Programm mit einem potentiellen Aliasing-Problem in der Testphase völlig problemlos, wenn die Aliasing-Bedingung sehr selten eintritt.
Zur Vermeidung von Konkurrenz- und Aliasing-Situationen müssen Threads an bestimmten kritischen Stellen synchronisiert werden, d.h. Threads müssen gezwungen werden, vor der weiteren Programmabarbeitung auf eine Art "Freizeichen" zu warten. Bei Verfügbarkeit des "Freizeichens" muß der wartende Thread sofort ein "Besetzt" signalisieren, um konkurrierende Threads zum Warten zu zwingen. Hat ein Thread den kritischen Programmbereich durchlaufen, muß er das "Besetzt" zurücknehmen und das "Freizeichen" signalisieren. Anderenfalls würden die auf das "Freizeichen" wartenden Threads auf ewig blockiert bleiben.
Bei genauer Überlegung der Problematik leuchtet ein, daß die Aktion "testen ob Freizeichen, wenn ja, dann besetzten" atomar sein muß, d.h. nicht durch einen Threadwechsel unterbrochen werden darf. Eine `wait-`Anweisung mit nachfolgendem Löschen einer "frei"-Variable erfüllt diese Voraussetzung nicht!
z.B.

```
wait free;   // 2 virtuelle Maschinenoperationen
free = 0;    // 2 virtuelle Maschinenoperationen
```

besteht aus 4 virtuellen Maschinenoperationen. Angenommen ein Thread bekommt hier das Freizeichen, dann erfolgt ein Threadwechsel, bevor er `free` auf 0 setzen kann. Ein zweiter Thread, der ebenfalls auf dieses Freizeichen wartet, erhält nun Rechenzeit. Das Signal steht noch auf "frei", obwohl schon ein anderer Thread den kritischen Bereich betreten hat!
Zur Lösung des Synchronisationsproblems stellt die virtuelle Maschine der C-Control II Unit eine atomare Maschinenoperation zur Verfügung, auf die in C2 über das Schlüsselwort **`capture`** zugegriffen werden kann.

Die **`capture`**-Anweisung existiert in einer expliziten und einer impliziten Form. Die explizite Form erwartet die Angabe eines Bezeichners einer globalen (!) byte-Variable.

z.B.:

```
byte flag;

thread tx
{
  capture flag;  // explicit: capture the flag

  //...
}
```

Beim Ausführen der **`capture`**-Anweisung testet der aktuelle Thread, ob der Wert der Variablen 0 ist (= "Freizeichen"). Wenn ja, dann schreibt der Thread seine eigene Nummer (Wert 1 … 255) in die Variable und merkt sich die Adresse der `byte`-Variable. Wie bereits erwähnt, dieser Vorgang läuft atomar, in einer einzigen Operation der virtuellen Maschine ab.

Die implizite Form lautet

```
capture;
```

ohne weitere Angaben. Die implizite Form kann nur innerhalb der Anweisungsblöcke von Funktionen verwendet werden, nicht in denen von Threads. Die implizite Form nutzt versteckte globale `byte`-Variablen, die der C2-Compiler automatisch anlegt. Zu jeder Funktion exisitiert eine solche Variable.

Das Setzen des "Freisignals" erfolgt durch das Schlüsselwort **release**. Da sich ein Thread merkt, welche byte-Variable er besetzt hat, steht die **release**-Anweisung ohne weitere Angaben.

```
release;
```

Ein Thread sollte **release** sofort aufrufen, wenn er den synchronisierten Bereich verläßt. Andere, vor **capture** wartende Threads werden sonst unnötig blockiert.

☞ Ein Thread darf niemals zwei capture-Anweisungen ohne zwischenzeitliches release ausführen.

Der C2-Compiler kann das nicht nachprüfen. Bei Mißachtung kommt es zu Blockaden im Programm.

z.B.

```
byte flag1;
byte flag2;

thread tx
{
  capture flag1;   // Blockade im 2. Durchlauf
  capture flag2;   // flag1 wird "vergessen"

  //...
  release;         // nur flag2 wird freigegeben
}
```

Der Thread tx im obigen Beispiel läuft wie jeder Thread automatisch in einer Endlosschleife. Im zweiten Schleifendurchlauf blockiert er, da flag1 noch auf "Besetzt" steht.
Abschließend je ein Beispiel zur Anwendung der expliziten und der impliziten **capture**-Form:

1.) Vermeidung von Aliasing bei Speicherzugriffen durch explizites **capture**

```
float a, b;  // Messwerte
byte flag;   // Synchronisationsvariable

thread measure
{
  capture flag;
  a = get_channel_a();
  b = get_channel_b();
  release;
  //...
}

thread watch
{
  capture flag;
  check(a, b);
  release;
  // ...
}
```

2.) Synchronisation von Ressourcenzugriffen durch implizites **`capture`** in einer Funktion

```
function send ( byte buf[], int length )
{
  capture;
  wait ressource.ready();
  ressource.send(buf, length);
  release;
}
```

Im Beispiel wird hier ein Modul `ressource` angenommen, dessen Funktionen selbst noch nicht synchronisiert sind. Alle Bibliotheksmodule zum Zugriff auf Systemressourcen der C-Control II Unit enthalten bereits die notwendige Synchronisation (siehe z.B. Modul hwcom.c2).

5.10 Anweisungen zur Ablaufsteuerung

Unentbehrlicher Teil einer strukturierten Programmiersprache sind Anweisungen zur Steuerung des Programmflusses. Erst dadurch können Algorithmen realisiert werden, die über die bloße rechnerische Verknüpfung von Werten hinausgehen.

5.10.1 if … else … - Bedingte Ausführung

Mit der **if-**Anweisung wird die Abarbeitung von Programmabschnitten an eine Bedingung geknüpft. Nach dem Schlüsselwort **if** folgt ein numerischer Ausdruck und danach eine Anweisung oder ein Anweisungsblock.

```
if Ausdruck Anweisung;
if Ausdruck
{
  //...
}
```

Die Anweisung bzw. der Block werden nur dann ausgeführt, wenn das Ergebnis des numerischen Ausdrucks zur Programmlaufzeit ungleich 0 ist.

z.B.

```
x = 123;

if x fx();      // fx wird aufgerufen
if x-123 fx();  // fx wird nicht aufgerufen
```

Über das Schlüsselwort **else** kann eine alternative Anweisung (oder ein Block) angegeben werden, die ausgeführt wird, wenn der Wert des Audrucks gleich 0 ist.

```
if Ausdruck
   Anweisung;
else
   AlternativAnweisung;
```

z.B.

```
x = 123;

if x-123  // ist 0
```

```
    fx1();
  else
    fx2();  // fx2 wird aufgerufen
```

5.10.2 loop - Endlosschleife

Programmschleifen ermöglichen das wiederholte Ausführen von Anweisungen. Die einfachste Form ist die bedingungslose Endlosschleife. Dafür kann in C2 das Schlüsselwort **loop** verwendet werden. Nach diesem steht eine einzelne Anweisung oder ein Anweisungsblock.

```
loop Anweisung;

loop
{
  //..
}
```

5.10.3 while - Schleife

Die **while**-Schleife wiederholt eine Anweisung oder einen Anweisungsblock, solange der berechnete Wert eines numerischen Bedingungsausdrucks ungleich 0 ist. Die Prüfung der Bedingung erfolgt vor jedem Schleifendurchlauf.
Nach dem Schlüsselwort **while** folgt ein numerischer Ausdruck und danach die Anweisung bzw. der Anweisungsblock.

```
while Ausdruck Anweisung;

while Ausdruck
{
  //..
}
```

5.10.4 do - Schleife

Die **do**-Schleife wiederholt eine Anweisung oder einen Anweisungsblock, solange der berechnete Wert eines numerischen Bedingungsausdrucks ungleich 0 ist. Die Prüfung der Bedingung erfolgt nach jedem Schleifendurchlauf. Die Anweisung der Schleife wird also mindestens einmal ausgeführt.

Nach dem Schlüsselwort **do** folgt die Anweisung bzw. der Anweisungsblock, danach das Schlüsselwort **while** und abschließend der numerische Bedingungsausdruck.

```
do Anweisung while Ausdruck;

do
{
  //...
}
while Ausdruck;
```

5.10.5 for - Schleife

Die **for**-Schleife führt eine Anweisung oder einen Anweisungsblock solange aus, bis eine Schleifenvariable eine Vergleichsbedigung nicht mehr erfüllt.

```
for Variable=Wert ... Operator Endwert step Wert
  Anweisung;

for Variable = Wert ... Operator Endwert step Wert
{
  //...
}
```

Nach dem Schlüsselwort **for** wird die Schleifenvariable mit einem Startwert initialisiert (*Variable=Wert*). Dann folgt das "**...**"-Symbol. Optional steht dann ein Vergleichsoperator (`Operator: <, >, <=` oder >=). Wird der Operator nicht angegeben, geht der C2-Compiler von einer "kleiner-gleich"-Bedingung aus (<=). Vor jedem Durchlauf wird die Schleifenvariable mit einem `Endwert` verglichen. Optional kann mit dem Schlüsselwort **step** eine Schrittweite angegeben werden. Ohne diese Angabe beträgt die Schrittweite konstant 1. Nach jedem einzelnen Durchlauf wird der Schrittweitenwert zur Schleifenvariable addiert.
Beispiele:

```
for i = 0 ... 10              // 11 Läufe
{
  //...
```

```
}

for i = 0 ... <10           // 10 Läufe
{
  //...
}

for i = 9 ... >=0 step -1   // 10 Läufe rückwärts
{
  //...
}
```

5.10.6 Vorzeitiger Abbruch

Wird eine `loop-`, `while-`, `do-` oder `for`-Schleife mit einem Anweisungsblock verwendet, kann es erwünscht sein, die Schleife unter bestimmten Sonderbedingungen vorzeitig abzubrechen - also ohne eventuelle weitere Anweisungen des Blocks auszuführen und ohne die Schleifenabbruchbedingung nochmals zu testen. Dafür kann die **`break`**-Anweisung benutzt werden.

```
break;
```

z.B.

```
for i = 0 ... 9999
{
  if bad() break;
  //...
}
```

5.10.7 Vorzeitige Fortsetzung

Wird eine `loop`-, `while`-, `do`- oder `for`-Schleife mit einem Anweisungsblock verwendet, kann es erwünscht sein, die Schleife unter bestimmten Sonderbedingungen vorzeitig mit dem nächsten Durchgang fortzusetzen, ohne eventuelle weitere Anweisungen des Blocks auszuführen.

```
continue;
```

z.B.

```
for i = 0 ... 9999
{
  if not (i mod 13) continue;

  //...
}
```

Im obigen Beispiel führen alle i, die ohne Rest durch 13 teilbar sind, zu einer Auslassung der Anweisungen, die ab //... folgen würden.

5.10.8 Programmende

Die gesamte Programmabarbeitung der virtuellen Maschine kann durch die **quit**-Anweisung beendet werden. Dazu muß nach dem **quit**-Schlüsselwort ein numerischer Ausdruck stehen, dessen berechneter Wert ungleich 0 ist. Mit diesem Wert kehrt das System in den Hostmodus zurück (siehe 4.3). Im Hostmodus könnte z.B. ein neues Programm von einem angeschlossenen PC übertragen und dann gestartet werden.

Wird quit mit dem Parameter -1 oder 255 aufgerufen führt das System einen Software-Reset des Mikrocontrollers aus.

```
quit Ausdruck;
```

z.B.

```
quit 63;   // zurück in den Hostmodus
quit -1;   // Sofware-Reset
```

6 Softwareentwicklung

6.1 Installation und Start der Integrierten Entwicklungsumgebung

Mit der C-Control II Unit haben Sie eine Utility-CD erhalten. Auf dieser CD befindet sich u.a. die Integrierte Entwicklungsumgebung, die Sie zur Programmierung der Unit benötigen. Die Integrierte Entwicklungsumgebung läuft unter den 32Bit-Betriebssystemen Microsoft Windows95/98/NT/2000.
Zur Installation legen Sie die CD in das CD-Laufwerk Ihres PC. Ist für das CD-Laufwerk der Autostart-Modus aktiviert (das ist die Standardeinstellung), dann öffnet sich nach Einlegen der CD ein Begrüßungsbildschirm. Folgen Sie bitte den Hinweisen dieses Begrüßungsbildschirms. Wenn Sie für Ihr CD-Laufwerk den Autostart-Modus deaktiviert haben, laden Sie bitte die Datei start.htm aus dem Wurzelverzeichnis der CD in Ihren aktuellen Internet-Browser. Wenn Sie noch keinen Internet-Browser installiert haben, lesen Sie bitte die Datei browser.txt im Wurzelverzeichnis der CD.
Nach erfolgreicher Installation der Integrierten Entwicklungsumgebung können Sie diese von Ihrem Windows-Startmenü aus starten. Die Benutzeroberfläche der Integrierten Entwicklungsumgebung entspricht aktuellen Standards und ist intuitiv zu bedienen. Details entnehmen Sie bitte der Online-Hilfe, die Sie durch Drücken der Taste [F1] aufrufen können.

6.2 Quelltexte bearbeiten

Im Editor der Integrierten Entwicklungsumgebung geben Sie die Quelltexte der einzelnen Module ein.
Durch das sogenannte Syntax Highlighting werden die verschiedenen Syntaxelemente in verschiedenen einstellbaren Farben und Schriftstilen angezeigt. Dadurch wird die Lesbarkeit der Programme am Bildschirm und beim Ausdruck erhöht. Außerdem können einfache Schreibfehler leichter erkannt werden. Der Editor verfügt über die üblichen Funktionen zum Laden und Speichern von Dateien, Suchen und Ersetzen von Textpassagen sowie über Undo und Redo für Quelltextänderungen.
Weitere Werkzeuge zur Unterstützung Ihrer Arbeit sind die automatische Vervollständigung von Bezeichnern während der Eingabe, kontextsensitive Hilfe zu C2-Schlüsselwörtern und modulübergreifendes Suchen nach der Definition von C2-Bezeichnern im aktuellen Projekt. Die genauen Beschreibungen dazu finden Sie in der Online-Hilfe der Integrierten Entwicklungsumgebung (Taste [F1]).

6.3 Richtlinien zur Quelltextformatierung

6.3.1 Vorteile der einheitlichen Formatierung

Für die syntaktische und funktionale Korrektheit eines Programms ist die Formatierung eines Quelltextes ohne Bedeutung. Im Interesse der Übersichtlichkeit und der Verständlichkeit sollten Quelltexte jedoch auch "optisch" korrekt sein. Ein stilvoll und diszipliniert gestalteter Quelltext nach einheitlichen Richtlinien ist auch nach längerer Zeit und auch für andere Programmierer lesbar und nachvollziehbar. Formatierte Quelltexte enthalten in der Regel von Anfang an weniger Fehler. Wenn sie Fehler enthalten, ist deren Suche und Beseitigung einfacher an einem sauberen Quelltext durchführbar.
Nachfolgende Richtlinien sind Gestaltungsvorschläge. Es steht Ihnen frei, die Vorschläge anzunehmen, zu variieren oder zu verwerfen. Wenn Sie jedoch Unterstützung von Conrad Electronic wünschen und dazu Quelltextauszüge zur Überprüfung einsenden, können diese nur bearbeitet werden, wenn sie den unten aufgeführten Richtlinien erkennbar entsprechen. Im Zweifel orientieren Sie sich bitte an der Formatierung der Standardmodule und Beispiele auf der CD zur C-Control II Unit.

6.3.2 Kommentare

1. Ein Programm soll Kommentare enthalten, wenn das zur wesentlichen Verbesserung der Verständlichkeit führt.
2. Ein Kommentar ist zu vermeiden, wenn der Sinn von Anweisungen auch durch selbstbeschreibende Bezeichner deutlich werden kann.
3. Kommentare sollen nicht trivial sein, z.B. `x = 1; // 1 an x zuweisen`
4. Zeilenendkommentare sollen zumindest für Folgen von Anweisungen, die nicht durch Leerzeilen getrennt sind, linksbündig untereinander stehen.
5. Ein erläuternder Kommentar zu einer Folge von Anweisungen steht in einer eigenen Zeile vor diesen Anweisungen, mit derselben Einrückung wie diese Anweisungen.
6. Kommentare sind in einer einzigen Sprache verfaßt, z.B. einheitlich englischsprachig oder einheitlich deutschsprachig. Kommentare sind in derselben Landessprache wie die Bezeichner zu formulieren.

6.3.3 Bezeichner

1. Bezeichner sollen selbstbeschreibend sein. Abkürzungen sind möglich, solange die Bedeutung im Kontext ohne zusätzliche Kommentare erkennbar bleibt. Zum Beispiel "getMaxTemp" statt "getMaximumTemperature" ist zulässig. Nur primitive Funktionen, temporäre Variablen zur Speicherung von Zwischenergebnissen, Indizes oder Schleifenvariablen, dürfen aus einzelnen Buchstaben oder kurzen Zeichenkombinationen bestehen.
2. Bezeichner von Modulen, Threads, Funktionen und Variablen beginnen mit einem Kleinbuchstaben.
3. Bezeichner von zusammengesetzten Datentypen beginnen mit einem Großbuchstaben.
4. Bezeichner von Konstanten bestehen nur aus Großbuchstaben, Unterstrichen und Ziffern.
5. Bezeichner von Funktionen sollten weitestgehend mit einem Verb oder einer üblichen Abkürzung eines Verbs beginnen (z.B get, set, put, write, init, calc …)
6. In längeren Bezeichnern sind einzelne Worte durch Unterstriche oder einzelne Großbuchstaben an Wortwechseln zu trennen, z.B. getMaxTemp oder get_max_temp. Die einmal gewählte Schreibweise ist beizubehalten.
7. Bezeichner sind in einer einzigen Sprache verfaßt, z.B. einheitlich englischsprachig oder einheitlich deutschsprachig. Bezeichner sind in derselben Landessprache wie die Kommentare zu formulieren.

6.3.4 Ausdrücke

Komplexe numerische Ausdrücke sind durch Klammersetzung und Leerzeichen so zu gestalten, daß Teilausdrücke optisch erkennbar sind.

6.3.5 Funktionsdefinitionen

1. Die Definitionszeilen sind durch "//-----" -Kommentarzeilen gerahmt (siehe Standardmodulquelltexte). Zwischen diesen Kommentarzeilen steht nichts außer der Funktionsdefinition.
2. Eine Definitionszeile beginnt mit einer Einrückung von zwei Leerzeichen.
3. Rechts und links der runden Klammern steht ein Leerzeichen, z.B. function fx (int param). Bei Funktionen ohne Parameter entfallen die Leerzeichen innerhalb der runden Klammern, z.B. function fx ().
4. Bei Funktionsdefinitionen mit mehreren Parametern steht nach jedem Komma in der Liste der formalen Parameter ein Leerzeichen, z.B. function fx (int a, int b, int c)

5. Erstreckt sich eine Definition über mehrere Zeilen, so sind die zweite und weitere Zeilen linksbündig unter dem Typen des ersten Parameters fortzusetzen, z.B.

```
//-------------------------------------------------
  function getMaxTemp ( int deviceID,
                        int channel ) returns int
//-------------------------------------------------
```

6.3.6 Threads

1. Die Definitionszeilen sind wie bei Funktionen durch "//-----" -Kommentarzeilen gerahmt.
2. Threads stehen stets am Ende eines Modulquelltextes.

6.3.7 Anweisungsblöcke

1. Die Definition lokaler Variablen ist durch eine Leerzeile von den restlichen Anweisungen getrennt.
2. Eine abschließende return-Anweisung ist durch eine Leerzeile von den vorangehenden Anweisungen getrennt.
3. Längere Anweisungsblöcke sind durch zusätzliche Leerzeilen geeignet zu strukturieren.
4. Geschweifte Klammern stehen paarweise untereinander.
5. Geschweifte Klammen stehen jeweils allein in einer Zeile.
6. Verschachtelte Anweisungsblöcke sind jeweils um zwei Leerzeichen eingerückt, z.B.

```
//-------------------------------------------------
  function getMaxTemp ( int channel ) returns int
//-------------------------------------------------
{
  int i;
  int result;

  if channel == 13
  {
    for i = 0...<20
    {
      // ...
    }
```

```
  }
  return result;
}
```

6.3.8 Kombinationen mit Schlüsselworten zur Ablaufsteuerung

1. Einzelne Anweisungen in Kombinationen mit Schlüsselworten zur Ablaufsteuerung (`if`, `else`, `loop`, `while`, `do`, `for`) stehen in derselben Zeile wie das Schlüsselwort oder vorzugsweise um zwei Leerzeichen eingerückt in der nächsten Zeile.
2. Die geschweiften Klammern von Anweisungsblöcken nach Schlüsselworten zur Ablaufsteuerung stehen linksbündig unter dem Schlüsselwort.
3. Vor den Schlüsselworten `if`, `loop`, `while`, `do` und `for` sollte eine Leerzeile stehen. Die Leerzeile kann entfallen, wenn es sich um verschachtelte Konstrukte handelt, und das Schlüsselwort eingerückt direkt nach einer `{`-Zeile folgt.

```
do fx() while not ready;

 do
  fx()
while not ready;

if x < LIMIT
  make_it();
else
  leave_it_alone();

while is_good()
{
  for x = 0...9
  {
    do_this();
    do_that();
  }
}
```

6.4 Automatischer Compiler

Bereits während der Eingabe des Quelltextes läuft im Hintergrund die Syntaxanalyse und Übersetzung durch den C2-Compiler. Im Ergebnis wird direkt im Editorfenster angezeigt, ob eine Programmzeile fehlerhaft ist (Kreuzsymbol) oder zu ausführbarem Code führt (Punktsymbol). Das entsprechende Symbol wird vor der Zeile angezeigt. Leere Zeilen oder solche, die nicht unmittelbar zu ausführbarem Code führen, haben kein Symbol. Wenn Ihre Eingabe Fehler enthält, werden Ihnen im Meldungsfenster konkrete Fehlerbeschreibungen angezeigt. Nachdem Sie alle Modulquelltexte eines Projektes vollständig geschrieben und alle syntaktischen Fehler beseitigt haben, kann Ihr Programm simuliert oder in die C-Control II Unit übertragen werden.

6.5 Simulation und Debugging

6.5.1 Test und Fehlersuche

Nachdem ein Programm syntaktisch korrekt compiliert wurde, muß die funktionelle Fehlerfreiheit überprüft werden. Es ist nicht ratsam, die C-Control II Unit mit einem Programm zu laden, dessen prinzipielle Funktion nicht im Simulator der Integrierten Entwicklungsumgebung getestet wurde. Schätzen Sie selbst ab, welche Folgen eine Programmfehlfunktion beim Betrieb Ihrer Applikation haben kann.
Von einfachsten Anwendungen abgesehen, wird ein Programm selten auf Anhieb so funktionieren, wie es im Detail gewünscht ist. Manche Fehlfunktion ist offensichtlich und reproduzierbar ("Immer wenn ich die Taste drücke, dann …"). Die Ursache kann meist leicht gefunden und beseitigt werden. Schwieriger ist das Finden von Fehlern, die nur in der Verkettung mehrerer, zum Teil seltener Bedingungen auftreten. ("Wochenlang läuft alles einwandfrei, dann …").

Einige Empfehlungen zum Test und zur Fehlersuche:

- Es gilt der Grundsatz: alles, was nicht getestet wurde, wird früher oder später Fehlfunktionen zeigen - niemals glauben, daß etwas funktioniert, sondern testen und wissen.
- Testen Sie ein Programm nicht erst im vollen Ausbau. Stellen Sie zunächst die Korrektheit aller einzelnen Unterprogramme (Threads, Funktionen) sicher, fügen Sie die Bestandteile stückweise zusammen, und führen Sie immer wieder Zwischentests durch.

- "Füttern" Sie Ihre Funktionen zum Test mit allen möglichen Eingabedaten, nicht nur mit den für Ihre Anwendung "normalen" Werten. Früher oder später kommt es zu "unnormalen" Situationen, an die Sie im Moment vielleicht nicht denken.
- Bauen Sie in Ihr Programm Statusausgaben auf das Mini-LCD ein, auch wenn Ihre Anwendung das LCD nicht benötigt, nutzen Sie freie Digitalports zur Ausgabe von Statussignalen. Beobachten Sie das Programmverhalten anhand der Statusausgaben und -signale.
- Kreisen Sie Fehler durch gezieltes Auskommentieren von Programmzeilen ein. Nutzen Sie Breakpoints, Einzelschrittbetrieb und die Überwachung und Anzeige von Variablen.

6.5.2 Simulationsumfang

Kern des Simulators ist dieselbe virtuelle Maschine, die auch im Betriebssystem der C-Control II Unit arbeitet. Die Ausführung aller Speicher-, Steuer-, und Rechenoperationen ist absolut identisch. Somit kann die logische und algorithmische Korrektheit eines Programmes getestet und sichergestellt werden.
Was der Simulator nicht oder nicht vollständig nachbildet, sind extern angeschlossene Systeme, z.B. ICs am I^2C-Bus oder Sensormodule. Außerdem entspricht das Timing des Simulators nicht dem der realen Unit. D.h. von der Ausführungszeit bestimmter Programmabschnitte am PC kann nicht auf das Zeitverhalten im Betrieb der C-Control II Unit geschlossen werden.
Daraus folgt:

☞ Die korrekte Funktion eines Programmes im Simulator ist ein notwendiges, jedoch kein hinreichendes Kriterium für den fehlerfreien Betrieb Ihrer Applikation in Echtzeit und unter realen Hardwarebedingungen!

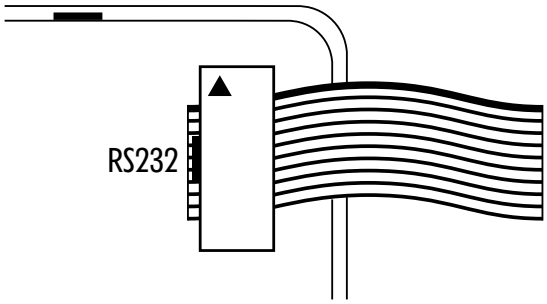
Im Simulator kann ein Programm im Einzelschrittmodus oder im ganzen gestartet werden. Dabei können Sie in speziellen Ausgabefenstern die Zustände der wichtigsten Hardwareressourcen (z.B. Digitalports) beobachten. Außerdem können Sie die Werte globaler und lokaler Varibalen anzeigen lassen.

6.5.3 Bedienung

Hinweise zur Bedienung des Simulators entnehmen Sie bitte der Online-Hilfe der Integrierten Entwicklungsumgebung.

6.6 Programmübertragung in die Unit

Schließen Sie die Control II Unit an einer seriellen Schnittstelle Ihres PCs an. Verwenden Sie dazu das der Unit beiliegende Nullmodem- und das SUB-D-Adapterkabel. Falls Sie kein Application Board oder eine andere Zwischenplatine benutzen, stecken Sie das Adapterkabel so auf die Pins von X1, daß die rote Ader des Flachbandkabels zum Gehäuserand zeigt:



Schließen Sie die Versorgungsspannung an die Unit an und aktivieren Sie den Hostmodus (siehe Kapitel 4.3).
Stellen Sie in der Integrierten Entwicklungsumgebung die korrekte Schnittstelle ein. Beenden Sie alle anderen Programme, die auf dieselbe serielle Schnittstelle zugreifen. Laden oder bearbeiten Sie ein C2-Projekt, compilieren Sie es und rufen Sie in der Integrierten Entwicklungsumgebung das Menü zum Übertragen des Programms auf. Lesen Sie dazu auch die Online-Hilfe.

7 Standardmodule

Dieses Kapitel gibt einen Überblick über alle Bibliotheksmodule zum Zugriff auf die Systemressourcen der C-Control II Unit. Im einzelnen sind das die Module

Datei	Inhalt
can.c2	CAN-Bus
constant.c2	allgemeine Konstanten
hwcom.c2	1. serielle Schnittstelle (Hardwareschnittstelle)
i2c.c2	I^2C-Bus
lcd.c2	Mini-LCD der Unit
lpt.c2	Druckerschnittstelle über digitale Ports der Unit
math.c2	Fließkomma-Arithmetik
mem.c2	Bytepufferoperationen
plm.c2	Pulsweitenmodulation für D/A-Wandlung und Tonausgabe
ports.c2	Digitalports und Analogports (A/D)
str.c2	Stringmanipulation
swcom.c2	2. serielle Schnittstelle (Softwareschnittstelle)
system.c2	Timer, Systemuhr, Interrupt-Umleitung
twb.c2	Zweidrahtbus
vmcodes.c2	Codekonstanten der virtuellen Maschine

7.1 can.c2 ☆☆☆

7.1.1 Initialisierung

```
function init ( int speed, int globalMask, int specialMask )
```

Vor der Datenübertragung auf dem CAN-Bus muß das System initialisiert werden. Der erste Parameter der `init`-Funktion dient zur Festlegung der Übertragungsgeschwindigkeit.

Folgende `speed`-Werte werden unterstützt.

speed	Übertragungsgeschwindigkeit
SPEED_50 (0)	50 kbit/s
SPEED_62 (1)	62,5 kbit/s
SPEED_125 (2)	125 kbit/s
SPEED_250 (3)	250 kbit/s
SPEED_500 (4)	500 kbit/s

Wenn Sie einen ungültigen Wert für `speed` übergeben, wird die Übertragungsrate auf 125 kbit/s festgesetzt.
Zur Akzeptanzfilterung eingehender CAN-Nachrichten müssen zwei Maskenwerte, `globalMask` und `specialMask`, spezifiziert werden. `globalMask` gilt für alle 15 Kanäle, für den 15. Kanal (channel 14) gilt zusätzlich die `specialMask`, die vom Mikrocontroller intern mit der `globalMask` UND-verknüpft wird. Die Akzeptanzmaske bestimmt, welche Bits der Message-ID einer eingehenden Nachricht mit der Empfangs-ID eines CAN-Kanals zu vergleichen sind, um zu entscheiden, ob eine eingehenden Nachricht für diesen Kanal bestimmt ist und zu empfangen ist. Ein 0-Bit bedeutet "don't care" (egal), ein 1-Bit bedeutet "compare" (muß verglichen werden).
Beispiel, gültig für alle Kanäle 0 … 13:

globalMask	0x0000	0x07FF	0x7FF	0x7FE
Empfangs-ID	egal	0x0120	0x120	0x120
Message-ID	egal	0x0120	0x121	0x121
Empfang ja/nein	ja	ja	nein	ja

Wenn `specialMask` ungleich `globalMask` ist, dann werden, wegen der UND-Verknüpfung, für Kanal 14 weniger Bits als bei den Kanälen 0 … 13 verglichen. Kanal 14 ist also für mehr eingehende Nachrichten empfangsbereit. Man könnte z.B. `globalMask` auf 0x07FF setzen (= alle 11 ID-Bits) und `specialMask` auf 0x0000. Dann ist jeder Kanal 0 … 13 nur für den Empfang genau einer Nachricht zuständig, und Kanal 14 ist ein Universalempfänger.

Weitere Details zum Nachrichtenempfang siehe auch ab 7.1.7.

7.1.2 Statusabfrage für einen CAN-Kanal

```
function ready ( int channel ) returns int
```

Die Funktion `ready` prüft, ob ein Kanal bereit für eine neue CAN-Übertragung ist.
`channel` 0 … 14 (channel 14 kann nur empfangen und ist nie bereit)
Rückgabe: -1 wenn bereit, sonst 0

7.1.3 Test auf Übertragungsfehler

```
function error () returns int
```

Die Funktion error befragt die integrierte CAN-Hardware des C164CI nach dem zuletzt aufgetretenen Fehler. Eine Zuordnung eines Fehlers zu einem einzelnen Kanal ist nicht möglich. Zum Verständnis der einzelnen Fehlercodes empfehlen wir dringend die Lektüre eines Fachbuches zum Thema CAN-Bus sowie der Systemdokumentation zum C164CI-Mikrocontroller.
Rückgabe:

Code-Konstante
ERROR_STUFF (1)
ERROR_FORM (2)
ERROR_ACK (3)
ERROR_BIT1 (4)
ERROR_BIT0 (5)
ERROR_CRC (6)
ERROR_EWRN (8)
ERROR_BUSOFF (9)

7.1.4 Nachricht senden

```
function send ( int channel, int id, byte buf[], int length )
```

Die Funktion `send` übergibt Bytes aus einem Bytepuffer an einen CAN-Ausgabekanal.

`channel` 0 … 13 (channel 14 kann nur empfangen!)
`id` Message-ID der Nachricht
`buf` Referenz auf Bytepuffervariable
`length` Pufferlänge, max. 8

7.1.5 Nachricht veröffentlichen

```
function publish ( int channel, int id, byte buf[],
                   int length )
```

Die Funktion `publish` übergibt Bytes aus einem Bytepuffer an einen CAN-Ausgabekanal und stellt die Daten für "Remote-Request"-Anforderungen anderer CAN-Busteilnehmer zur Verfügung. D.h. andere Busteilnehmer können unter Angabe der passenden Message-ID die Übertragung der Pufferdaten anfordern.

`channel`	0 … 13 (channel 14 kann nur empfangen!)
`id`	Message-ID der Nachricht
`buf`	Referenz auf Bytepuffervariable
`length`	Pufferlänge, max. 8

7.1.6 Zählen der "Remote-Request"-Anfragen

```
function rtrcount ( int channel ) returns byte
```

Die Funktion `rtrcount` liefert nach Veröffentlichung einer Nachricht einen Zählerwert, wie oft diese Nachricht von anderen Busteilnehmern abgefragt wurde. Der Zähler ist jedoch auf den Wertebereich eines Bytes beschränkt. Wird eine Nachricht öfter als 253 mal abgefragt, bleibt der Zählerwert auf 253 stehen.

`channel`	0 … 13 (channel 14 kann nur empfangen!)

7.1.7 Einstellen der Empfangs-ID

```
function expect ( int channel, int id )
```

Für jeden Kanal, der zum Empfangen von CAN-Nachrichten benutzt werden soll, muß eine Empfangs-ID eingestellt werden (siehe auch 7.1.1). Werden für mehrere Kanäle gleiche Empfangsbedingungen hergestellt, resultierend aus der Akzeptanzmaske und der Empfangs-ID, so wird eine eingehende Nachricht, die diesen Bedingungen entspricht, im niedrigsten freien Kanal gespeichert. Ein Kanal ist frei, wenn seine zuletzt empfangene Nachricht mit `get` ausgelesen wurde (siehe 0).

`channel`	0 … 14
`id`	Empfangs-ID des Kanals

7.1.8 Senden einer "Remote-Request"-Anforderung

```
function request ( int channel )
```

So wie die C-Control II Unit Nachrichten veröffentlichen kann (siehe 7.1.5), kann sie auch selbst eine Nachricht anfordern, die ein anderer CAN-Busteilnehmer veröffentlicht hat. Es muß bekannt sein, unter welcher Message-ID diese Nachricht abrufbar ist. Diese ID muß zuvor per `expect` für den Kanal `channel` als Empfangs-ID eingestellt sein, sonst kann die Antwort des Busteilnehmers nicht empfangen werden.

7.1.9 Test auf Empfang

```
function rxd ( int channel ) returns int
```

Die Funktion `rxd` testet, ob eine neue Nachricht auf einem Empfangskanal `channel` verfügbar ist. Wenn das so ist, gibt sie den Wert -1 zurück, anderenfalls 0.

`channel` 0 … 14

7.1.10 Empfangene Daten lesen

```
function get ( int channel, byte buf[] ) returns int
```

Daten, die auf einem Kanal `channel` automatisch oder nach einem `request` empfangen wurden, können mit `get` abgeholt und in eine Bytepuffervariable übertragen werden. Der Puffer muß Platz für 8 Bytes bieten. Die Funktion liefert als Ergebnis die Anzahl der Bytes, die tatsächlich empfangen wurden; gültige Nachrichten können auch aus 0 Datenbytes bestehen.

`channel` 0 … 14
`buf` Referenz auf Bytepuffervariable

7.2 hwcom.c2 und swcom.c2

Die C-Control II -Unit verfügt über zwei asynchrone serielle Schnittstellen. Eine davon ist als Hardware (hwcom) bereits im Mikrocontroller implementiert. Die zweite Schnittstelle (swcom) kann vom Betriebssystem softwaremäßig über zwei interruptsensible Ports nachgebildet werden. Der Zugriff auf beide Schnittstellen in C2 ist identisch. Nachfolgend

beschriebene Funktionen sind in den Modulen hwcom.c2 und swcom.c2 gleichartig definiert.

7.2.1 Initialisierung

```
function init ()
```

Die Funktion `init` initialisiert eine serielle Schnittstelle und deaktiviert eventuell konkurrierende Portfunktionen.

7.2.2 Einstellen der Übertragungsgeschwindigkeit

Für jede der beiden Schnittstellen kann die Übertragungsgeschwindigkeit eingestellt werden. Die hwcom-Schnittstelle kann dabei bis 115.200 Baud arbeiten, swcom nur bis 9.600 Baud.

```
function setspeed ( int speed )
```

speed	Übertragungsgeschwindigkeit
SPEED_300 (0)	300 Baud
SPEED_600 (1)	600 Baud
SPEED_1200 (2)	1200 Baud
SPEED_2400 (3)	2400 Baud
SPEED_4800 (4)	4800 Baud
SPEED_9600 (5)	9600 Baud
SPEED_19200 (6)	19200 Baud
SPEED_38400 (7)	38400 Baud
SPEED_57600 (8)	57600 Baud
SPEED_115200 (9)	115200 Baud

(SPEED_19200 bis SPEED_115200 nur für hwcom)

7.2.3 Setzen des erweiterten Empfangspuffers

Das Betriebssystem implementiert für beide seriellen Schnittstellen standardmäßig je einen Empfangspuffer von 64 Byte. In Applikationen, in denen größere Datenblöcke zu empfangen sind, sollte ein erweiterter Empfangspfuffer reserviert werden. Anderenfalls kann es zum Verlust empfangener Daten führen, die vom Programm nicht schnell genug aus dem Puffer gelesen werden.

```
function setbuf ( byte buf[], int length )
```

`buf`	Referenz auf eine statische oder quasi-statische Bytepuffervariable
`length`	Länge des Puffers

7.2.4 Verwerfen von Daten

Applikationen, die serielle Daten in Rahmenform empfangen, können unter bestimmten Bedingungen unvollständige Rahmen im Empfangspuffer enthalten. Dann ist es erforderlich, alle Bytes im Empfangspuffer zu verwerfen, um auf den Beginn des nächsten Rahmens zu synchronisieren.

```
function flush ()
```

Die Funktion flush entfernt alle Daten aus dem Empfangspuffer.

7.2.5 Test auf Empfang

```
function rxd () returns int
```

Die Funktion `rxd` testet, ob ein oder mehrere neue Bytes im Empfangspuffer einer seriellen Schnittstelle verfügbar sind. Wenn das so ist, gibt sie den Wert -1 zurück, anderenfalls 0.

7.2.6 Lesen eines empfangenen Bytes

```
function get () returns byte
```

Die Funktion `get` liest und entfernt ein einzelnes Byte aus dem Empfangspuffer.

7.2.7 Empfang von Datenrahmen

```
function receive ( byte buf[], int length, long timeout )
                  returns int
```

Die Funktion `receive` liest und entfernt eine Anzahl von Bytes (Datenrahmen) aus dem Empfangspuffer und kopiert diese in eine Bytepuffervariable. Enthält der Empfangspuffer bei Aufruf der Funktion weniger empfangene Bytes als spezifiziert, wartet die Funktion auf

den Empfang weiterer Bytes. Das Warten wird abgebrochen, wenn zwischen zwei Bytes eine längere Pause erkannt wird (`timeout`). Der Rückgabewert gibt die Anzahl der tatsächlich gelesenen Bytes zurück.

`buf`	Referenz auf eine Bytepuffervariable
`length`	Länge des Puffers
`timeout`	Timeout in Millisekunden

7.2.8 Test auf Sendebereitschaft

```
function ready () returns int
```

Die Funktion `ready` prüft, ob eine serielle Schnittstelle bereit für eine neue Übertragung ist. Rückgabe: -1 wenn bereit, sonst 0

7.2.9 Senden eines Bytes

```
function put ( byte c )
```

Die Funktion `put` sendet ein einzelnes Byte über eine serielle Schnittstelle.

7.2.10 Senden von Datenrahmen

```
function send ( byte buf[], int length)
```

Die Funktion `send` sendet eine Anzahl von Bytes über die serielle Schnittstelle. Das Senden erfolgt im Hintergrund. D.h. die Funktion übergibt lediglich die Adresse des Sendepuffers an das System und startet die Übertragung. Daher muß die Bytepuffervariable statisch sein (globale Variable oder Variable eines Threads).

`buf`	Referenz auf eine statische Bytepuffervariable
`length`	Länge des Puffers

7.3 i2c.c2

Über die Funktionen des Moduls i2c.c2 kann ein Programm auf ICs zugreifen, die am I²C-Bus der C-Control II Unit angeschlossen sind. Eine typische Anwendung ist der Anschluß serieller EEPROMs zur Aufzeichnung von Daten.
Der I²C-Bus der C-Control II Unit ist als Single-Master-Bus implementiert. D.h. die Taktsignale, die Start- und Stopbedingungen werden stets von der C-Control II Unit erzeugt.

7.3.1 Initialisierung

```
function init ()
```

Die Funktion `init` initialisiert den I²C-Bus. Da auch das LCD der C-Control II Unit am I²C-Bus angeschlossen ist, wird die Initialisierung bereits automatisch beim Reset des Systems durchgeführt und muß in der Regel nicht im Anwenderprogramm vorgenommen werden.

7.3.2 Start der Übertragung

```
function start ( byte device ) returns int
```

Jeder Zugriff auf ein IC am I²C-Bus erfolgt durch das Erzeugen der Startbedingung mit anschließendem Senden der Geräteadresse auf den Bus. Die Funktion `start` übernimmt diese Aufgabe. Welche Geräteadresse für welches IC welche Bedeutung hat, entnehmen Sie bitte der Dokumentation zu diesen ICs.
Das Ergebnis der Funktion `start` ist -1, wenn das angesprochene IC bereit ist, anderenfalls 0.

7.3.3 Senden der Stopbedingung

```
function stop ()
```

Eine Datenübertragung auf dem I²C-Bus wird durch die Stopbedingung abgeschlossen. Die Funktion `stop` erzeugt dieses Signal auf dem Bus.

7.3.4 Schreiben eines Bytes

```
function write ( byte c ) returns int
```

Die Funktion `write` sendet ein Byte auf dem I²C-Bus. Das Ergebnis der Funktion ist -1, wenn das angesprochene IC mit einem Acknowledge geantwortet hat, anderenfalls 0.

7.3.5 Lesen eines Bytes mit Acknowledge

```
function read () returns byte
```

Die Funktion `read` liest ein Byte über den I²C-Bus und antwortet mit einem Acknowledge-Signal. Eine typische Anwendung ist das sequentielle Lesen von Bytes aus einem seriellen EEPROM.

7.3.6 Lesen eines Bytes ohne Acknowledge

```
function readlast () returns byte
```

Die Funktion `readlast` liest ein Byte über den I²C-Bus und antwortet mit einem No-Acknowledge-Signal. Eine typische Anwendung ist das Lesen des letzten Bytes einer Bytesequenz aus einem seriellen EEPROM.

7.3.7 Test auf Sendebereitschaft

```
function ready () returns int
```

Die Funktion `ready` prüft, ob der I²C-Bus bereit für eine neue Übertragung ist.
Rückgabe: -1 wenn bereit, sonst 0

7.4 *(entfällt)*

7.4.1 *(entfällt)*

7.4.2 *(entfällt)*

7.4.3 *(entfällt)*

7.4.4 *(entfällt)*

7.4.5 *(entfällt)*

7.4.6 *(entfällt)*

7.4.7 *(entfällt)*

7.4.8 *(entfällt)*

7.4.9 *(entfällt)*

7.4.10 *(entfällt)*

7.5 lpt.c2

Die Digitalports der C-Control II Unit können u.a. als parallele Druckerschnittstelle benutzt werden.

7.5.1 Initialisierung

```
function init ()
```

Die Funktion `init` initialisiert die Druckerschnittstelle. Eventuell konkurrierende Portfunktionen werden deaktiviert.

7.5.2 Ausgabepuffer leeren

```
function flush ()
```

Die Funktion `flush` leer den Ausgabepuffer und beendet so einen im Hintergrund laufenden Druckvorgang.

7.5.3 Test auf Ausgabebereitschaft

```
function ready () returns int
```

Die Funktion `ready` prüft, ob die Druckerschnittstelle bereit für eine neue Ausgabe ist.
Rückgabe: -1 wenn bereit, sonst 0

7.5.4 Ein Zeichen drucken

```
function put ( byte c )
```

Die Funktion `put` gibt ein einzelnes Zeichen (ASCII-Code) auf dem Drucker aus.

7.5.5 Text drucken

```
function print ( byte buf[], int length )
```

Die Funktion `print` gibt Zeichen (ASCII-Codes) aus einer Bytepuffervariable auf dem Drucker aus.
Die Ausgabe erfolgt im Hintergrund. D.h. die Funktion übergibt lediglich die Adresse des Ausgabepuffers an das System und startet die Übertragung. Daher muß die Bytepuffervariable statisch sein (globale Variable oder Variable eines Threads).

`buf`	Referenz auf eine statische Bytepuffervariable
`length`	Länge des Puffers

7.6 math.c2

7.6.1 Mathematische Standardfunktionen

Die Definitionen der Standardfunktionen haben die Form

```
function fx ( float x ) returns float
```

Name	berechnetes Ergebnis
sqr	Quadrat
sqrt	Quadratwurzel
curt	Kubikwurzel
sin	Sinus, Argument x in Bogenmaß (Radiant)
cos	Cosinus, Argument x in Bogenmaß (Radiant)
tan	Tangens, Argument x in Bogenmaß (Radiant)
asin	Arkussinus, Ergebnis in Bogenmaß (Radiant)
acos	Arkuscosinus, Ergebnis in Bogenmaß (Radiant)
atan	Arkustangens, Ergebnis in Bogenmaß (Radiant)
log	Logarithmus zur Basis 10
ln	natürlicher Logarithmus
exp	Exponentialfunktion e^x
ceil	kleinste ganze Zahl, die nicht kleiner als x ist (Aufrunden)
floor	größte ganze Zahl, die nicht größer als x ist (Abrunden)

7.6.2 Potenzieren

Die Funktion

```
function pow ( float x, float y ) returns float
```

berechnet die y-Potenz zur Basis x (x y, "x hoch y").

7.6.3 Absolutwertfunktionen

Für jeden numerischen Datentyp (außer `byte`, das entspricht `int` beim Funktionsaufruf) gibt es eine Absolutwertfunktion:

```
function abs ( int value ) returns int
function labs ( long value ) returns long
function fabs ( float value ) returns float
```

7.6.4 Minimum- und Maximumfunktionen

Für jeden numerischen Datentyp (außer `byte`, das entspricht `int` beim Funktionsaufruf) gibt es eine Minimum- und eine Maximumfunktion:

```
function min ( int a, int b ) returns int
function lmin ( long a, long b ) returns long
function fmin ( float a, float b ) returns float
function max ( int a, int b ) returns int
function lmax ( long a, long b ) returns long
function fmax ( float a, float b ) returns float
```

7.7 mem.c2

Die Funktionen des Moduls mem.c2 ermöglichen verschiedene Manipulationen an Bytepuffervariablen. Hauptanwendungen dieser Funktionen sind

- der Aufbau von Datenrahmen vor einer Datenübertragung
- das Lesen von Daten aus empfangenen Datenrahmen
- die Zeilenformatierung vor einer Ausgabe, z.B. auf einem Drucker

7.7.1 Füllen mit einem Wert

```
function fill ( byte buf[], int length, byte c )
```

Die Funktion `fill` füllt eine angegebene Bytepuffervariable mit einer Anzahl gleicher Zeichen, z.B. Leerzeichen.

`buf` Referenz auf eine Bytepuffervariable
`length` Füll-Länge
`c` Zeichen (ASCII-Code)

7.7.2 Kopieren

```
function copy ( byte dest[], int pos, byte src[],
                int length )
```

Die Funktion `copy` kopiert eine Anzahl (`length`) Zeichen aus einer Bytepuffervariable (`src`) an eine bestimmte Position (`pos`) einer anderen Bytepuffervariable (`dest`). Es ist darauf zu achten, daß der Zielpuffer genügend Platz für die kopierten Zeichen bietet.

7.7.3 Speichern von Zahlenwerten in einem Bytepuffer

Beim Aufbau von Datenpuffern vor einer Übertragung müssen oft Zahlenwerte gespeichert werden, die mehr Platz als je ein einzelnes Byte benötigen: Integer-, Long-, oder Fließkommawerte. Dazu können folgende Funktionen des Moduls mem.c2 benutzt werden:

```
function putint ( byte dest[], int pos,
                  int value )

function putlong ( byte dest[], int pos,
                   long value )

function putfloat ( byte dest[], int pos,
                    float value )
```

Für alle drei Funktionen ist

`dest` Referenz auf eine Bytepuffervariable
`pos` Ausgabeposition im Puffer
`c` der Zahlenwert

Die Funktion `putint` speichert den Wert in zwei Bytes ab der Position `pos` in der Folge HiByte - LoByte;
`putlong` speichert den Wert in vier Bytes ab der Position `pos` in der Folge HiByte des HiWord - LoByte des HiWord - HiByte des LoWord - LoByte des LoWord.
`putfloat` speichert den Wert in acht Bytes ab der Position `pos` im IEEE-Format ab. Da dieses Format nicht von allen Computersystemen gleichermaßen interpretiert wird, sollte `putfloat` nur für den Datenaustausch zwischen C-Control II -Systemen verwendet werden.

7.7.4 Lesen von Zahlenwerten aus einem Bytepuffer

Zu den Funktionen zum Schreiben von Zahlenwerten gibt es je eine entsprechende Funktion zum Lesen der Werte aus einem Bytepuffer.

```
function getint ( byte src[], int pos )
                  returns int

function getlong ( byte src[], int pos )
                   returns long

function getfloat ( byte src[], int pos )
                    returns float
```

Für alle drei Funktionen ist

`src` Referenz auf eine Bytepuffervariable
`pos` Leseposition im Puffer

7.8 plm.c2

7.8.1 Setzen der Zeitbasis

Es können acht veschiedene Zeitbasiswerte eingestellt werden. Das Einstellen erfolgt mit der Funktion `settimebase`.

```
function settimebase ( int channel, int timebase )
```

`channel` PLM-Kanal (0, 1, 2)
`timebase` Zeitbasis

Die übergebenen Zahlenwerte ergeben folgende Zeitbasen:
(Beachten Sie bitte, daß channel 0 und channel 1 eine gemeinsame Zeitbasis haben.)

timebase	Zeitbasis (Dauer eines Ticks)
BASE_400 (0)	400 ns
BASE_800 (1)	800 ns
BASE_1600 (2)	1,6 µs
BASE_3200 (3)	3,2 µs
BASE_6400 (4)	6,4 µs
BASE_12800 (5)	12,8 µs
BASE_25600 (6)	25,6 µs
BASE_51200 (7)	51,2 µs

7.8.2 Setzen des Portmodus

Jeder der drei PLM-Ports kann in einem von zwei verschiedenen Hardwaremodi betrieben werden: entweder mit digitalem Ausgangspegel oder mit Transistor-Push-Pull-Ausgang. Das Einstellen des Modus erfolgt mit der Funktion `setmode`.

```
function setmode ( int channel, int mode )
```

`channel` PLM-Kanal (0, 1, 2)
`mode` Portmodus (0 = digital, 1 = push-pull Transistorstufe)

7.8.3 Einstellen der Periodenlänge

Das Einstellen der Periodenlänge erfolgt mit der Funktion

```
function setperiod ( int channel, int length )
```

`channel` PLM-Kanal (0, 1, 2)
`length` Periodenlänge, N Ticks

Beachten Sie bitte, daß channel 0 und channel 1 eine gemeinsame Periodenlänge haben.

7.8.4 PLM-Ausgabe

```
function out ( int channel, int value )
```

`channel` PLM-Kanal (0, 1, 2)
`length` Periodenlänge, N Ticks

Die Funktion `out` gibt einen Wert pulsweitenmoduliert an einem PLM-Port aus. Ist der Ausgabewert mindestens so groß wie die für diesen Kanal eingestellte Periodenlänge, so ist der Ausgangspegel des Ports permanent high. Ein Ausgabewert 0 führt zu permanentem Lowpegel.

7.8.5 Ausgabe von Tonfrequenzen

An jedem der drei PLM-Ports kann über eine bestimmte Periodenlänge und einen Ausgabewert von z.B halber Periodenlänge ein Rechtecksignal mit einer bestimmten Frequenz ausgegeben werden. Dabei ist die eingestellte Zeitbasis zu berücksichtigen. Die Ausgabefrequenz der Pulslängenmodulation für einen Kanal ergibt sich aus 1 / (Zeitbasis * Periodenlänge), wenn der PLM-Ausgabewert kleiner als die Periodenlänge und größer als 0 ist.
Zur Vereinfachung der Ausgabe von Tonfrequenzen gibt es die Funktion beep. Sie bezieht sich stets auf den dritten PLM-Kanal (channel 2). Alle Berechnungen und Einstellungen von Periodenlängen unter Berücksichtigung der aktuellen Zeitbasis übernimmt diese Funktion.

```
function beep ( int tone )
```

Der Parameter `tone` bestimmt einen Ton im Bereich der Töne a bis c''''. Ein Ton mit der Frequenz von 440Hz ist der Kammerton a' - der Ton einer Stimmgabel.

Ein negativer `tone`-Wert schaltet den Ton ab und legt den PLM-Port auf konstanten Lowpegel. Eine Anzahl von `tone`-Konstanten ist in der Moduldatei plm.c2 definiert. Vor der Benutzung von `beep` sollte eine niedrige Zeitbasis gewählt werden, da dadurch die Tonfrequenzen mit höherer Präzision wiedergegeben werden.

7.9 ports.c2

Die C-Control II Unit stellt insgesamt 16 Digitalports und 8 A/D-Wandlerports des Mikrocontrollers an ihren Pins zur universellen Anwendung bereit. Der Zugriff auf diese Ports erfolgt über Funktionen des Moduls ports.c2.

Zwischen den digitalen Prozessorports und den Portnummer-Parametern der Funktionen dieses Moduls besteht folgender Zusammenhang:

Prozessor-Port	Nummern der Einzelports	Nummern der Nibbleports	Nummern der Byteports
P1L.0	0	0	0
P1L.1	1		
P1L.2	2		
P1L.3	3		
P1L.4	4	1	
P1L.5	5		
P1L.6	6		
P1L.7	7		
P1H.0	8	2	1
P1H.1	9		
P1H.2	10		
P1H.3	11		
P1H.4	12	3	
P1H.5	13		
P1H.6	14		
P1H.7	15		

7.9.1 Abfrage von Digitalports

Digitalports können einzeln, in Vierergruppen (Nibbles), byteweise und im Ganzen als ein 16bit-Integer (Word) abgefragt werden. Das Ergebnis der Abfrage ist immer ein Integerwert, der als Bitmaske den Portzustand widerspiegelt: 1-Bit = Port high; 0-Bit = Port low. Beachten Sie folgende Besonderheit:

☞ Die Abfrage eines einzelnen Digitalports liefert - wie eine Vergleichsoperation - das Ergebnis -1 (Port ist high) oder 0 (Port ist low).

Parameter der Abfragefunktionen ist die Nummer des Ports, der erste Port hat jeweils die Nummer 0.

```
function get ( int number ) returns int
function getn ( int number ) returns int
function getb ( int number ) returns int
function getw ( int number ) returns int
```

Folgende Tabelle zeigt gültige Portnummern und den Wertebereich der Ergebnisse der einzelnen Abfragefunktionen.

Funktion	Anwendung	Portnummern	Ergebnis
`get`	Abfrage von Einzelports	0 … 15	0, -1
`getn`	Abfrage von Nibbleports	0 … 3	0 … 15
`getb`	Abfrage von Byteports	0 … 1	0 … 255
`getw`	Abfrage von Wordports	0	0x0000 … 0xFFFF

7.9.2 Setzen von Digitalports

Jeder der 16 Digitalports kann als Eingang oder als Ausgang benutzt werden. Für die Anwendung als Ausgang muß vor der ersten Ausgabe die interne Elektronik des Mikrocontrollers entsprechend aktiviert werden. Das übernimmt das Betriebssystem der C-Control II Unit automatisch beim Aufruf der set…-Funktionen. Das Setzen von Ports kann wie beim Lesen einzeln, als Nibble, als Byte oder als 16bit-Integer (Word) erfolgen. Parameter der set...-Funktionen sind die Portnummer und der zu setzende Portzustand als Bitmaske. Höherwertige Bits, die in einer Ausgabe nicht darstellbar sind, werden ignoriert; z.B. das Setzen eines Nibbleports auf den Wert 17 (0b10001) ist nicht möglich und wird als Setzen auf 1 (0b00001) interpretiert. Bei der Ausgabe auf Einzelports führen alle Werte ungleich 0 zum Setzen des Ports auf Highpegel.

```
function set ( int number, int state )
function setn ( int number, int state )
function setb ( int number, int state )
function setw ( int number, int state )
```

7.9.3 Umschalten und Pulsen

Nachdem ein Port mit einer set…-Funktion initialisiert wurde, stehen folgende Funktionen zur Verfügung:

```
function toggle ( int number )
function pulse ( int number )
```

Die Funktion `toggle` invertiert einen Port. Die `pulse`-Funktion gibt einen Nadelpuls an einem Port aus (zweimaliges Invertieren kurz hintereinander). Das kann zum Beispiel als clock-Signal für digitale Schaltkreise mit Triggereingang benutzt werden.

Beide Funktionen beziehen sich jeweils auf einen einzelnen Digitalport, dessen Nummer als Parameter übergeben wird.

7.9.4 Deaktivieren von Ports

Wird ein Digitalport nach Aufruf einer set…-Funktion als Ausgang betrieben, sind im Mikrocontroller spezielle Transistorstufen aktiviert, die am Port einen Strom treiben (Port high) oder gegen Masse ziehen können (Port low). In manchen Anwendungen sollen Digitalports als Ausgang und dann wieder als Eingang betrieben werden. Dazu müssen die Treiberstufen abgeschaltet - deaktiviert - werden. Das erfolgt durch Aufruf folgender Funktionen, jeweils für einen Einzelport, Nibbleport, Byteport oder Wordport:

```
function deact ( int number )
function deactn ( int number )
function deactb ( int number )
function deactw ( int number )
```

7.9.5 Pulszählung

```
getcount ( int number ) returns long
```

Die vier Digitalports P1H.0 … P1H.3 sind interruptsensibel. Sie werden vom Betriebssystem beim Reset so eingerichtet, daß sie eingehende Inpulse zählen (bei High-Low-Flanke am Digitalport). Diese vier Zählerstände können über die Funktion `getcount` abgefragt werden. Als Parameter erwartet die Funktion die Portnummer 0 … 3. Bei jeder Abfrage eines Zählerstandes wird dieser auf 0 zurückgesetzt. Das Aufsummieren über einen größeren Zeitraum muß im Anwenderprogramm erfolgen.

7.9.6 Frequenzmessung

```
function getfreq ( int number ) returns long
```

Die Pins DCF/FRQ 0 und FRQ 1 der C-Control II Unit können zur Messung von Pulsfrequenzen benutzt werden. Die Abfrage erfolgt mit der Funktion `getfreq`, mit der Nummer 0 oder 1 als Parameter.

7.9.7 Analog-Digital-Wandlung

```
function adc ( int number ) returns int
```

Die Funktion `adc` liefert den digitalisierten Meßwert von einem der 8 ADC-Ports der C-Control II Unit. Die Nummer des Ports (0 … 7) wird als Parameter übergeben. Das Ergebnis ist ein Integer im Bereich von 0 bis 1023 - entsprechend der 10bit-Auflösung des A/D-Wandlers des Mikrocontrollers; siehe dazu auch Kapitel 3.2.3.

7.10 str.c2

Die C-Control II Unit unterstützt einfache Stringoperationen bereits auf Ebene der virtuellen Maschine. Der Zugriff auf diese Operationen erfolgt über Funktionen des Moduls str.c2.

7.10.1 String leeren

```
function clear ( string s )
```

Die Funktion `clear` leert den als Referenz übergebenen String, seine Länge wird auf 0 gesetzt.

7.10.2 Stringlänge ermitteln

```
function length ( string s ) returns int
```

Die Funktion `length` ermittelt die Länge des als Referenz übergebenen Strings.

7.10.3 String mit Zeichen füllen

```
function fill ( string s, int pos, int c )
```

Die Funktion `fill` füllt einen String `s` ab der Position `pos` (0…29) bis zur maximalen Länge (30 Zeichen) mit dem Zeichen `c` (ASCII-Code).

7.10.4 Ausgabe in einen String

Über die `put`...-Funktionen im Modul str.c2 können Inhalte an eine existierende Stringvariable angehängt werden: einzelne Zeichen, Teilstrings, Integer-, Long- oder Floatwerte:

```
function putchar ( string s, int c )
function putstring ( string dest, string source )
function putint ( string s, int value )
function putlong ( string s, long value )
function putfloat ( string s, float value )
```

7.10.5 Formatierte Ausgabe in einen String

C2 unterstützt einige einfache Formatierungen beim Anhängen von Zahlenwerten an Strings. Das Format wird durch einen Integerparameter bestimmt.

· int-Ausgabe

```
function putintf ( string s, int value, int format )
```

Die Funktion `putintf` hängt einen formatierten Integer an einen String an. Der Parameter `format` legt die Anzahl der Ausgabestellen fest. Fehlende führende Stellen werden mit "0" aufgefüllt.

```
putintf (s, 1, 4);
```

hängt also "0001" an s an. Negative `format`-Werte führen zur Ausgabe als Hexadezimal-Zahl mit Großbuchstaben.

```
putintf (s, 255, -4);
```

erweitert s um "00FF".

· long-Ausgabe

```
function putlongf ( string s, long value,
                    int format )
```

Die Funktion `putlongf` arbeitet identisch zu `putintf`, sie akzeptiert jedoch einen Long-Wert zur Ausgabe.

· float-Ausgabe

```
function putfloatf ( string s, float value,
                     int format )
```

Bei der Funktion `putfloatf` legt der `format`-Parameter die Anzahl der Nachkommastellen fest. Gegebenenfalls wird eine Anzahl von Nullen hinter dem Dezimalpunkt ausgegeben.

z.B.

```
putfloatf(s, 1, 3);
```

hängt "1.000" an `s` an. Insgesamt werden maximal 8 Ziffern (vor und nach dem Dezimalpunkt) ausgegeben.

7.10.6 Ausgabe einer Bitmaske

```
function putmask ( string s, int value,
                   int c1, int c0 )
```

Die Funktion putmask gibt einen Integerwert (0…255) als 8 Bitzeichen in einen String aus. Das Zeichen für Highbits wird durch den Parameter c1 bestimmt (ASCII-Code), c0 legt das Lowzeichen fest.

z.B.

```
putmask(s, 170, 'o', '-');
```

hängt "o-o-o-o-" an `s` an. Mit `putmask` können zum Beispiel Byteport-Zustände einfach zur Ausgabe auf dem LCD vorbereitet werden.

7.11 system.c2

7.11.1 Systemtimer

Das Betriebssystem verwaltet einen freilaufenden Timer. In jeder Millisekunde wird der Timer um 1 erhöht. Der Zählerstand wird vom System in einer internen long-Variable gespeichert. Der aktuelle Wert dieser Variablen kann durch die Funktion

```
function timer () returns long
```

abgefragt werden. Beachten Sie, daß der Zählerstand gemäß dem Wertebereich von long-Variablen nach 2147483647 in den negativen Wert -2147483648 überläuft.

7.11.2 Uhrzeit

Die C-Control II Unit verfügt über eine interne Echtzeituhr, die sich durch den Anschluß einer DCF77-Aktivantenne sekundengenau synchronisieren kann. Darüber hinaus kann die Uhrzeit auch im C2-Programm gestellt werden, und zwar durch Aufruf der Funktion

```
function settime ( int hour, int minute, int second )
```

Die Teilwerte der aktuellen Uhrzeit (Stunde, Minute, Sekunde) können durch folgende Funktionen abgefragt werden:

```
function hour () returns int
function minute () returns int
function second () returns int
```

Beachten Sie, daß zwischen den einzelnen Abfragen eine neue Minute oder Stunde anbrechen kann. Beutzen Sie daher vorzugsweise die Funktion

```
function gettime ( TIME time )
```

Diese Funktion gibt die volle Uhrzeit im Block in die als Parameter übergebene Datenstruktur vom Typ TIME aus:

```
type TIME
{
  int hour;
  int minute;
  int second;
}
```

7.11.3 Status der DCF77-Synchronisation

Das Betriebssystem versucht zu jeder vollen Minute, die interne Echtzeituhr auf den empfangenen DCF77-Datenrahmen zu synchronisieren. Unter schlechten Empfangsbedingungen kann eine Synchronisation über einen längeren Zeitraum ausfallen. Die interne Echtzeituhr läuft dann quarzgetaktet weiter. Bedingt durch Temperatureinflüsse und Toleranzen der elektronischen Bauteile führt das nach einer längeren Zeit zu einer zunehmenden Zeitabweichung der internen Uhr. Um im C2-Programm die Aktualität und Genauigkeit der internen Uhr abzuschätzen, kann über die Funktion `dcferr` ein Zähler des Betriebssystems abgefragt werden, der die Anzahl der vergeblichen Synchronisationen wiedergibt.

```
function dcferr () returns int
```

Das Rücksetzen des Zählers erfolgt mit jeder korrekten Synchronisation. Ist dann z.B. innerhalb von 30 Minuten keine neue Synchronisation möglich, steht der Zähler auf 30. Bei dauerhaftem Synchronisationsausfall wird der Zähler auf dem Wert 32767 festgehalten. Auch beim Reset wird der Zähler mit diesem Wert initialisiert.

7.11.4 Datum

Mit der DCF77-Synchronisation wird auch das Datum des Systems gestellt. Wenn Sie das Datum im C2-Programm manipulieren möchten, benutzen Sie die Funktion

```
function setdate ( int year, int month, int day )
```

Zur Abfrage der einzelnen Datumsinformationen dienen die Funktionen

```
function year () returns int
function month () returns int
function day () returns int
function dow () returns int
```

`dow` liefert den Wochentag. Dabei steht 0 für Sonntag, 1 für Montag usw. bis 6 für Samstag. Stellen Sie in Ihrem Programm vor der Abfrage der einzelnen Datumswerte sicher, daß nicht zwischendurch ein Tageswechsel (Mitternacht) auftreten kann. (vgl. Quellcode der Funktion `gettime`).

7.11.5 Sommerzeitflag

Die Funktion

```
function dst () returns int
```

gibt die Information zurück, ob es sich beim aktuellen Systemdatum um ein Datum in der Sommerzeitperiode handelt: 0 entspricht der Normalzeit (Winterzeit), -1 bedeutet Sommerzeit.

7.11.6 Aufruf von Systemfunktionen ☆☆☆

Aus C2-Programmen heraus können beliebige Funktionen des Betriebssystems oder anwenderdefinierte Assembler-/C-Routinen aufgerufen werden. Dazu gibt es im Modul system.c2 die Funktion

```
function call ( int segment, int offset )
```

Über den Aufruf der Funktion

```
function jump ( int segment, int offset )
```

können Sie die virtuelle Maschine der C-Control II Unit und somit die Abarbeitung eines C2-Programms verlassen und zu einer beliebigen Routine im Gesamtadreßraum des C164CI springen.

Voraussetzung für den Aufruf von `call` und `jump` ist, daß Sie jeweils die Adresse (`segment, offset`) der Funktion kennen. Die Adressen Ihrer eigenen Assembler-/C-Routinen entnehmen Sie bitte den Ausgaben Ihrer Assembler-/C-Entwicklungstools. Ein Beispiel zur Anwendung von call finden Sie auf der Utility-CD.

7.11.7 Anwenderdefinierte Interruptroutinen ☆☆☆

Zur unverzögerten Reaktion auf die Ereignisse

- 1ms-Timerzyklus des Systems,
- High-Low-Flanken an den Digitalports P1H.0 … P1H.3,

können Interruptroutinen in Assembler oder C geschrieben und im Segment 3 des FLASH-EEPROMs gespeichert werden. Lesen Sie dazu das Kapitel "8 Systemprogrammierung". Das Aktivieren der anwenderdefinierten Interruptroutinen erfolgt durch Aufruf der Funktion `hook`.

```
function hook ( int event, int segment, int offset,
                int mode )
```

Die Funktion `hook` "hängt" eine anwenderdefinierte Interruptroutine in die normale Interruptbehandlung des Systems ein. Der Parameter `event` gibt vor, für welche Interruptquelle eine Interruptroutine aktiviert werden soll:

event	Interruptquelle
`EVENT_TIMER (0)`	1 ms Timer
`EVENT_P1H0 (1)`	Digitalport P1H.0
`EVENT_P1H1 (2)`	Digitalport P1H.1
`EVENT_P1H2 (3)`	Digitalport P1H.2
`EVENT_P1H3 (4)`	Digitalport P1H.3

Die Parameter `segment` und `offset` geben die Speicheradresse der Interruptroutine im Gesamtadreßraum des Mikrocontrollers an. Die Speicheradresse entnehmen Sie bitte den Ausgaben Ihrer C-/Assembler-Entwicklungstools. Lesen Sie dazu die Dokumentation zu diesen Tools.
Wenn Sie eine C-Funktion als Interruptroutine schreiben möchten, muß sie im Stil

```
void fx ( void )
```

definiert sein, also ohne Parameter und Rückgabewert.
Wird eine Interruptroutine in die normale Interruptbehandlung des Systems eingehängt, gibt es für die Abarbeitung bei Auftreten des Interrupts drei Möglichkeiten. Die

gewünschte Variante bestimmen Sie durch den Parameter `mode`:

mode	Ausführung der anwenderdefinierten Interruptroutine
HOOK_REPLACE (0)	an Stelle der normalen Interruptbehandlung des Systems
HOOK_BEFORE (1)	vor der normalen Interruptbehandlung des Systems
HOOK_AFTER (2)	nach der normalen Interruptbehandlung des Systems

☞ Das Aktivieren eigener Interruptroutinen stellt einen erheblichen Eingriff in das Gesamtsystem dar und hat entscheidenden Einfluß auf dessen Zeitverhalten! Interruptroutinen müssen so kurz wie möglich gehalten werden.

Eine Interruptroutine für ein Ereignis kann durch Aufruf der Funktion `unhook` deaktiviert werden

```
function unhook ( int event )
```

7.12 twb.c2

7.12.1 Initialisierung

Die Initialisierung der 2W-Bus-Schnittstelle erfolgt mit der Funktion

```
function init ()
```

Eventuell konkurrierende Portfunktionen werden deaktiviert.

7.12.2 Abfrage auf Empfang des Antwortrahmens

Die Funktion

```
function rxd () returns int
```

liefert -1, wenn ein Antwortrahmen vom 2W-Bus-Modem empfangen wurde, anderenfalls 0.

7.12.3 Datenübertragung

Die Kommunikation mit den 2W-Bus-Modulen läuft stets über ein 2W-Bus-Modem. Zwischen der C-Control II Unit und dem Modem werden seriell-synchron 8 Byte lange

Datenrahmen übertragen. Die Bedeutung der einzelnen Bytes entnehmen Sie bitte den Anleitungen zu den 2W-Bus-Modulen und dem Modem.

```
function io ( byte buf[] ) returns int
```

Die Funktion `io` erwartet als Parameter eine Referenz auf ein 8 Byte langes Array. Ihr Programm muß die an den 2W-Bus zu sendenden Informationen dort eintragen, z.B. die Adresse des angesprochenen Moduls, das Kommando und eventuelle Datenbytes. Die Funktion `io` überträgt diesen Bytepuffer und wartet auf den Empfang des Antwortrahmens. Dieser Antwortrahmen wird in den übergebenen Bytepuffer `buf` übertragen.

Der Rückgabewert der Funktion ist -1 bei erfolgreicher Datenübertragung und 0 im Fehlerfall (das Modul hat nicht geantwortet).

Beachten Sie bitte, daß Sie in jedes 2W-Bus-Modul vor der eigentlichen Anwendung eine eindeutige Adresse übertragen müssen. Lesen Sie dazu die Anleitungen zu den 2W-Bus-Modulen und dem Modem. Einige dieser Anleitungen hatten ihren Redaktionsschluß vor Erscheinen der C-Control II Unit. Der Text der Anleitungen enthält daher keine expliziten Hinweise auf C-Control II.

7.13 constant.c2 und vmcodes.c2

Das Modul `constant` enthält einige allgemeine Konstanten, die im Quelltext selbst betrachtet werden können.
Das Modul `vmcodes` listet alle Operationscodes der virtuellen Maschine auf. Eine Dokumentation der Codes ist nicht Bestandteil dieser Anleitung und ist zur Anwendungsprogrammierung der C-Control II Unit nicht erforderlich.

8 Systemprogrammierung ☆☆☆

8.1.1 TASKING C/C++ Tools

Das Betriebssystem der C-Control II Unit wurde mit der Vollversion der TASKING C/C++ Tools entwickelt. Eine Demoversion dieser Tools finden Sie auf der Utility CD zur C-Control II Unit. Diese Tools enthalten u.a. eine Entwicklungsumgebung mit Editor und Projektverwaltung, einen integrierten C/C++-Compiler, einen Assembler und Linker. Nähere Informationen entnehmen Sie bitte den Dateien und Installationshinweisen auf der CD.

8.1.2 Ergänzungen der virtuellen Maschine und Änderungen am Betriebssystem

In das Segment 3 des externen FLASH-EEPROMs der Mikrocontrollerschaltung können Sie kleine Systemroutinen zur Ergänzung der virtuellen Maschine laden. Diese können aus C2-Programmen heraus mit den Funktionen `system.call` und `system.jump` aufgerufen oder per Funktion `system.hook` als Interrupt-Handler für Digitalports oder den Systemtimer installiert werden (siehe Kapitel 7.11).

Die Routinen in Assembler, C oder C++ können mit Hilfe der Demoversion der Tasking C/C++ Tool übersetzt werden. Beachten Sie dabei die Limitierungen der Demoversion. Das erzeugte Ausgabefile im Intel-Hexformat läßt sich mit Hilfe der C2- Entwicklungsumgebung in das Segment 3 der Unit übertragen. Die Adressen Ihrer Funktionen finden Sie in der erzeugten Map-Datei.

Wenn Sie aus Ihren Assembler-, C- oder C++-Routinen auf Daten und Funktionen des Betriebssystems zugreifen müssen, oder Sie das von Conrad Electronic ausgelieferte Betriebssystem nach Ihren Wünschen anpassen möchten, benötigen Sie die Quelltexte und die Dokumentation des Systems sowie einen geeigneten Compiler. Die Quelltexte und deren Dokumentation erhalten Sie auf Anfrage als Sonderbestellung bei Conrad Electronic. Einen passenden Compiler finden Sie in der Vollversion der TASKING C/C++ Tools. Aktuelle Informationen zu Preisen und Support finden Sie ab Verfügbarkeit auf der C-Control Homepage www.c-control.de.

8.1.3 Implementierung eines eigenen Betriebssystems

Prinzipiell können Sie ein vollständig eigenes Betriebssystem entwerfen und in die C-Control II Unit laden. Sie sollten dazu über umfangreiche Kenntnisse in der Anwendung und Programmierung des C164CI-Mikrocontrollers verfügen. Außerdem benötigen Sie eine geeignete Entwicklungsumgebung, z.B. die Vollversion der TASKING C/C++ Tools. Bitte haben Sie Verständnis, daß wir für die Programmierung Ihrer eigenen Betriebssysteme keinen kostenfreien Support leisten können. Sollten Sie Interesse an einer Entwicklungsunterstützung haben, vermitteln wir Ihnen gerne ein entsprechendes Angebot.

9 Anhang

9.1 Technische Daten

Hinweis: detailliertere Informationen finden Sie in den PDF-Dateien der IC-Hersteller auf der C-Control -Utility CD.
Alle Spannungsangaben beziehen sich auf Gleichspannung (DC).

9.1.1 Mechanik

äußere Abmessungen ohne Pins	ca. 82mm x 60mm x 18 mm
Masse	ca. 90g
Pinraster	2,54mm

9.1.2 Umgebungsbedingungen

Bereich der zulässigen Umgebungstemperatur	0°C … 40°C
Bereich der zulässigen relativen Umgebungsluftfeuchte	20% … 60%

9.1.3 Versorgungsspannung

Bereich der zulässigen Versorgungsspannung	8V … 24V
Stromaufnahme der Unit ohne externe Lasten	ca. 60 … 100mA
max. zulässiger Dauerstrom aus der stabilisierten 5V-Spannung der Unit	100mA
max. zulässige Verlustleistung der Unit	1,5W

9.1.4 Ports

max. zulässiger Strom aus digitalen Ports	± 5 mA
max. zulässige Summe der Ströme an digitalen Ports	50 mA
Bereich zulässiger Eingangsspannung an den Portpins (digital und A/D)	-0,5V … 5,5 V

9.2 Literaturverzeichnis

SIEMENS (Infineon): "16Bit-Microcontrollers - C164CI User's Manual"

R. Schultes, I. Pohle: 80C166 Mikrocontroller, Franzis' Verlag, 1994, ISBN 3-7723-5893-4

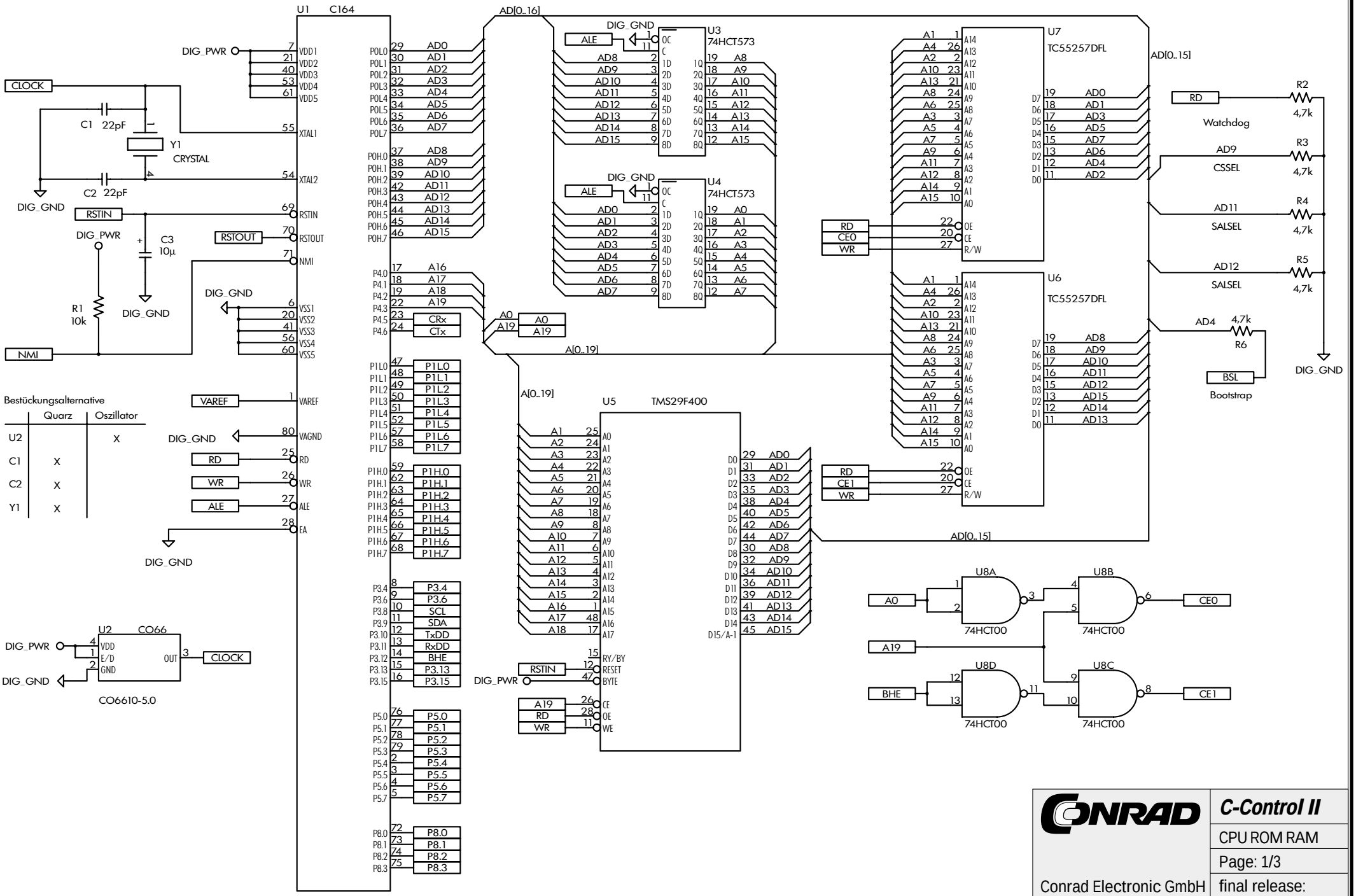
W. Lawrenz: "CAN Controller Area Network, Grundlagen und Praxis", 2. Auflage, Hüthig Verlag, 1997, ISBN 3-7785-2575-1

K. Dembowski: "Computerschnittstellen und Bussysteme", Hüthig Verlag, 1997, ISBN 3-7785-2526-3

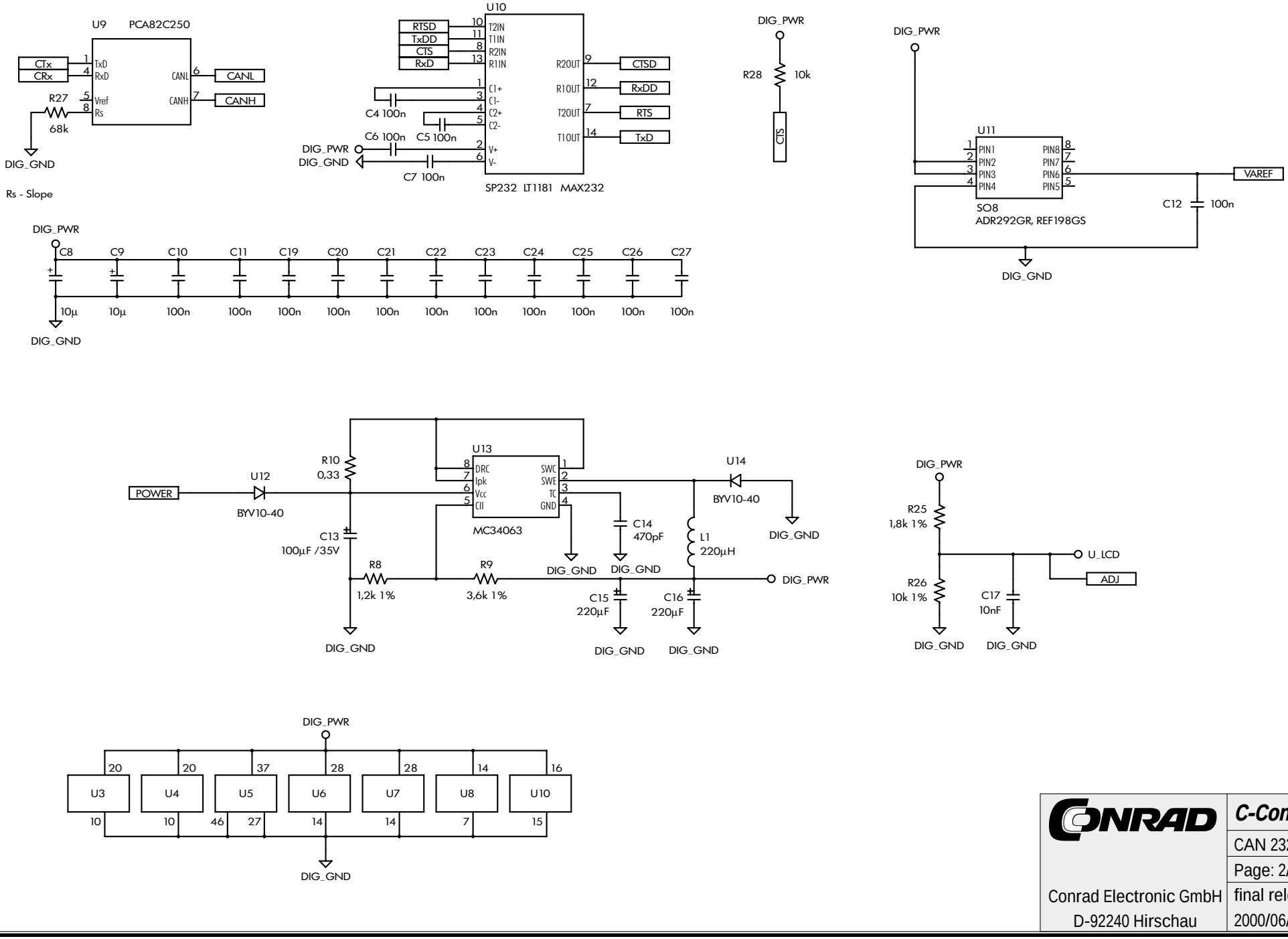
P. Rechenberg, G. Pomberger (Hrsg.): "Informatik-Handbuch", Carl Hanser Verlag, 1997, ISBN 3-446-18691-3

S. McConnell: "Code Complete", Microsoft Press, 1994, ISBN 3-86063-333-3

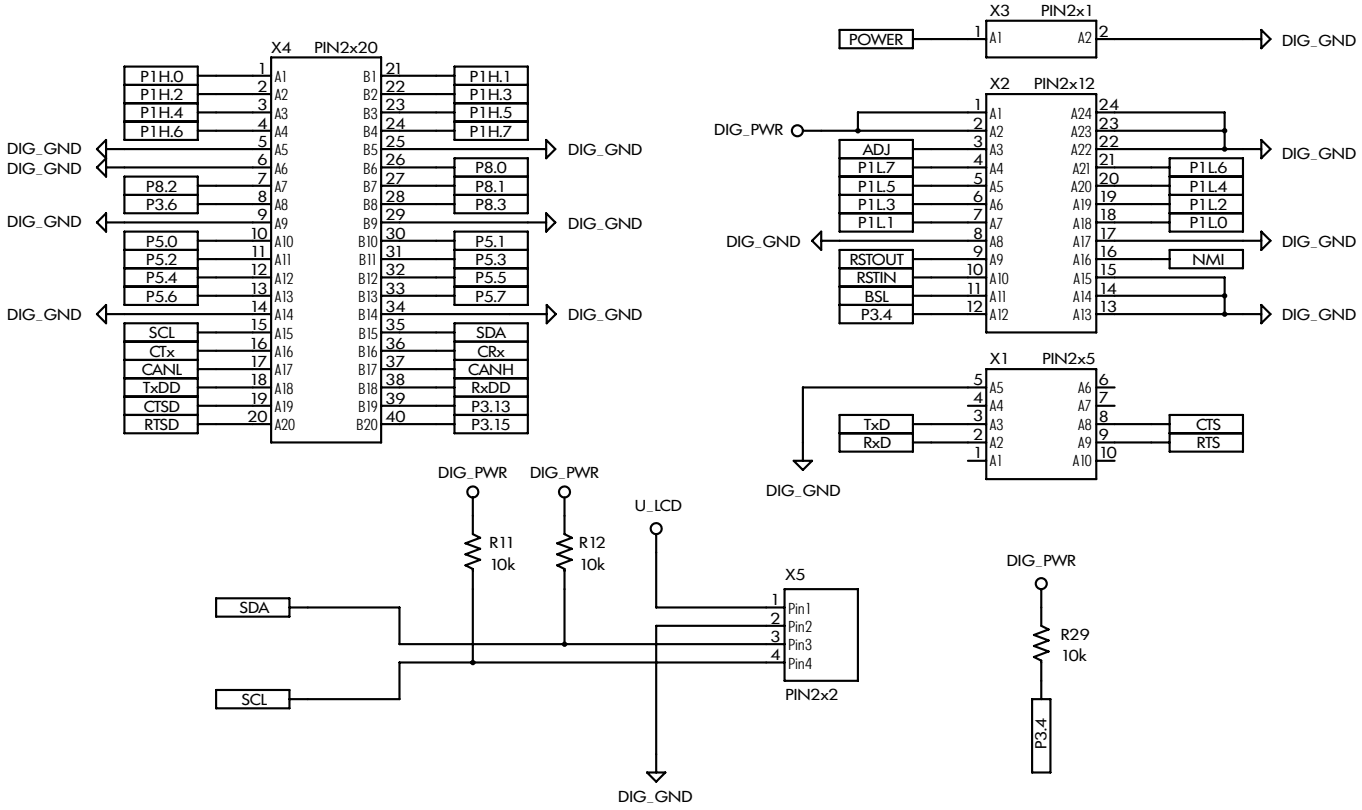
R. Hopfer, R. Müller: "BASIC Einführung in das Programmieren", VEB Fachbuchverlag Leipzig, 1987, ISBN 3-343-00367-0 ;-)



Conrad Electronic GmbH D-92240 Hirschau	**C-Control II** CPU ROM RAM Page: 1/3 final release: 2000/06/02



CONRAD	*C-Control II*
	CAN 232
	Page: 2/3
Conrad Electronic GmbH D-92240 Hirschau	final release: 2000/06/02



CONRAD	*C-Control II*
	Anschlüsse
	Page: 3/3
Conrad Electronic GmbH D-92240 Hirschau	final release: 2000/06/02

Detaillierte Definitionen und Anwendungsbeispiele finden Sie im Verlauf dieser Anleitung.

C2 Schlüsselworte	Seite		Seite		Seite		Seite
and	64	function	66	or	64	step	84
break	85	halt	74	quit	86	string	49
byte	49	if	82	release	80	thread	72
capture	79	inline	72	resume	75	type	49
const	59	int	48	return	69	wait	76
continue	85	long	48	returns	66	while	83
do	83	loop	83	run	74	xor	64
else	82	nand	64	shl	62	yield	75
float	48	nor	64	shr	62		
for	84	not	63	sleep	76		

Rangfolge der Operatoren in C2: (S. 61)	
Rang	Operator
8	( )
7	- (negatives Vorzeichen) ! not
6	* / % mod
5	+ -
4	<< shl >> shr
3	== != > < >= <=
2	& and !& nand
1	| or !| nor ^ xor

Arithmetische Operatoren (S. 61)	
Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
% mod	Modulodivision (Divisionsrest)
-	negatives Vorzeichen

Bitschiebeoperatoren (S. 62)	
Operator	Bedeutung
<< shl	links schieben
>> shr	logisch rechts schieben

Vergleichsoperatoren (S. 62)	
Operator	Bedeutung
==	ist gleich?
!=	ist ungleich?
>	ist göβer?
<	ist kleiner?
>=	ist größer oder gleich?
<=	ist kleiner oder gleich?

Logische Operatoren und Bitmanipulationen (S. 63)	
Operator	Bedeutung
! not	nicht (Bitinvertierung)
& and	und
!& nand	und mit anschließender Bitinvertierung
| or	oder
!| nor	oder mit anschließender Bitinvertierung
^ xor	exklusiv-oder

Detailierte Informationen zur Anschlußbelegung finden Sie in Kapitel 3.3 ab Seite 22

X4

P1H.0	P1H.1
P1H.2	P1H.3
P1H.4	P1H.5
P1H.6	P1H.7
GND	GND
GND	PLM 0
BEEP	PLM 1
FRQ 1	DCF/FRQ 0
GND	GND
ADC 0	ADC 1
ADC 2	ADC 3
ADC 4	ADC 5
ADC 6	ADC 7
GND	GND
I²C SCL	I²C SDA
CAN TxD	CAN RxD
CANL	CANH
digital TxD	digital RxD
digital CTS	µC CTS
digital RTS	µC RTS

X3

POWER	GND

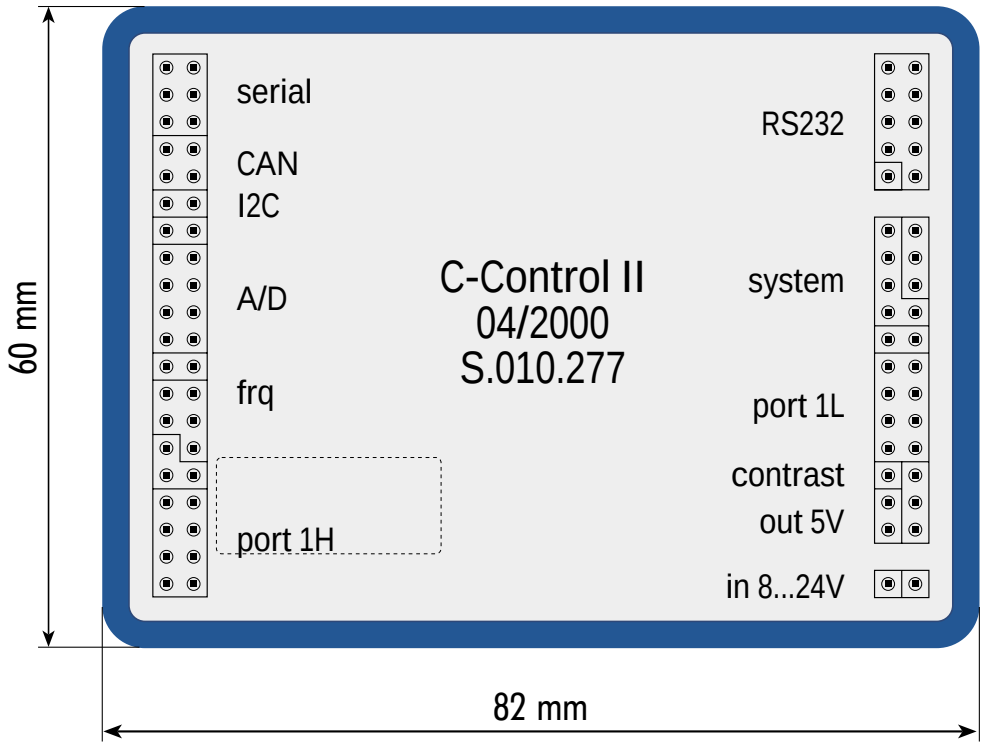
X2

5 Volt	GND
5 Volt	GND
LCD ADJ	GND
P1L.7	P1L.6
P1L.5	P1L.4
P1L.3	P1L.2
P1L.1	P1L.0
GND	GND
RSTOUT	NMI
RSTIN	GND
BOOT	GND
HOST	GND

X1

GND	n.c.
n.c.	n.c.
TxD	CTS
RxD	RTS
n.c.	n.c.

Raster 2,54 mm



LCD Zeichensatz